CS 201, Summer 2015 — August 13th — Final Exam Name: Solution

Question 1. [5 points] What output is printed by the following program (which begins on the left and continues on the right)?

```
public class Q1 {
  public static void f(
        String a, String b) {
    String temp = a;
    a = b;
    b = temp;
  }

  public static void main(
        String[] args) {
    String p = "Boy";
    String q = "Howdy";
    f(p, q);
    System.out.println(p + " " + q);
  }
}
```

Boy Howdy

Question 2. [5 points] What output is printed by the following program (which begins on the left and continues on the right)?

```
public class Q2 {
  public String s;

  public Q2(String s)
    { this.s = s; }

  public static void g(
        Q2 a, Q2 b) {
    String tmp = a.s;
    a.s = b.s;
    b.s = tmp;
  }

  public static void main(
        String[] args) {
    Q2 x = new Q2("Oh");
    Q2 y = new Q2("Yeah");

    g(x, y);
    System.out.println(
       x.s + " " + y.s);
  }
}
```

Yeah Oh

Question 3. [5 points] What output is printed by the following program (which begins on the left and continues on the right)?

```
public class Q3 {
  public static void f(
      int[] a, int [] b) {
    int[] tmp = a;
    a = b;
    b = tmp;
  }
```

```
  public static void main(
      String[] args) {
    int[] x = new int[]{1, 2};
    int[] y = new int[]{3, 4};
    f(x, y);
    System.out.printf("%d %d\n",
      x[0], y[0]);
  }
}
```

1 3

Question 4. [5 points] Complete the following static method. It should return the index of the *last* occurrence of `val` in the given array (`arr`). As a special case, if `arr` does not contain any elements equal to `val`, it should return -1.

Here are some example JUnit tests (which assume that the `findLastOccurrence` method is in a class called `Q4`):

```
String[] sarr = new String[]{"A", "B", "A", "C", "A", "B"};
assertEquals(5, Q4.findLastOccurrence(sarr, "B"));
assertEquals(4, Q4.findLastOccurrence(sarr, "A"));
assertEquals(3, Q4.findLastOccurrence(sarr, "C"));
assertEquals(-1, Q4.findLastOccurrence(sarr, "D"));
```

Hint: Think about how to compare `val` to the elements of the array.

```
public static<E> int findLastOccurrence(E[] arr, E val) {
  for (int i = arr.length - 1; i >= 0; i--) {
    if (arr[i].equals(val)) {
      return i;
    }
  }
  return -1;
}
```

Question 5. [5 points] Consider the following method:

```
public static int countLinesInFile(String fileName) {
  try {
    FileReader fr = new FileReader(fileName);
    BufferedReader br = new BufferedReader(fr);
    int count = 0;
    while (br.readLine() != null) {
      count++;
    }
    br.close();
    return count;
  } catch (IOException e) {
    return -1;
  }
}
```

try { ... } finally { br.close(); }

(a) Explain how it is possible that this method might open a file without closing it.

The call to readLine() could throw an IOException, so the call to close() would never be reached

(b) Explain how to modify the method so that the file is guaranteed to be closed (if it is opened).

Add try/finally as shown above

Question 6. [5 points] What output is printed by the following code?

```
Integer a = new Integer(42);
Integer b = new Integer(42);

if (a == b) {
    System.out.println("first");
}
if (a.equals(b)) {
    System.out.println("second");
}
```

second

Question 7. [5 points] What is the big-O upper bound on the worst-case running time of the following method? The problem size N is the length of the array passed as a parameter to the method. Explain your answer briefly.

```
public static void int mystery(int[] a) {
    int sum = 0;

    for (int j = 0; j < a.length; j++) {
        for (int i = 0; i < a.length; i++) {
            if (a[i] * a[j] == 1000000) {
                return -1;
            } else {
                sum += a[i] * a[j];
            }
        }
    }
    return sum;
}
```

N times
N times
O(1)

$N \cdot N \cdot O(1)$ is $O(N^2)$

Question 8. [5 points] Consider the following method (which begins on the left and continues on the right:

```
public static void prependEvens(
    List<Integer> list) {
  List<Integer> evens =
    new LinkedList<Integer>();

  // get all even values
  // and put them in evens list
  for (Integer v : list) {
    if (v % 2 == 0) {
      evens.add(v);
    }
  }

  // prepend the even values
  // to the list
  for (Integer e : evens) {
    // add at index 0, prepending e to
    // become the first element of list
    list.add(0, e);
  }
}
```

O(N)

N/2 times

O(1) for LinkedList,
O(N) (avg.) for
ArrayList

Let the problem size N be the number of elements in the parameter (`list`).

(a) What is the big-O upper bound of the running time if the list parameter is an instance of `ArrayList`? Explain briefly.

$O(N^2)$: second loop executes $N/2$ iterations, and call to add takes time proportional to # of elements in list

(b) What is the big-O upper bound of the running time if the list parameter is an instance of `LinkedList`? Explain briefly.

$O(N)$: second loop executes $N/2$ iterations, and call to add is $O(1)$ for a LinkedList

Question 9. [5 points] Consider the following classes:

```
public class IntBox {
  private int val;

  public IntBox(int val)
    { this.val = val; }

  public int getVal()
    { return val; }
}
```

```
public class IntPair
       extends IntBox {
  private int val2;

  public IntPair(
       int val, int val2) {
    TODO
  }

  public int getVal2() {
    return val2;
  }
}
```

Below, specify what code could be substituted for `TODO` so that the assertions in the following JUnit test code will succeed:

```
IntPair p = new IntPair(17, 42);
IntPair p2 = new IntPair(23, 79);

assertEquals(17, p.getVal());
assertEquals(42, p.getVal2());

assertEquals(23, p2.getVal());
assertEquals(79, p2.getVal2());
```

Note that you are only specifying statements to replace `TODO`. You may not modify either class in any way.

```
super(val);
this.val2 = val2;
```

Question 10. [5 points] Complete the following static method. It should return the number of elements of the given list which compare as either greater than the value `max` or less than the value `min`. The element type `E` is guaranteed to implement the `Comparable` interface.

The following JUnit test demonstrates the method:

```
List<Integer> list =
    Arrays.asList(41, 33, 26, 19, 34, 32, 32, 44, 19, 10);

// Count number of elements greater than 35 or less than 15
// (should count just the elements 41, 44, and 10)
int count = Q10.countElementsOutsideRange(list, 35, 15);
assertEquals(3, count);
```

Hints:

- Think about how the `Comparable` interface will help you compare the list elements to the `max` and `min` values

```
public static<E extends Comparable<E>>
int countElementsOutsideRange(List<E> list, E max, E min) {
    int count = 0;
    for (E val : list ) {
        if (val.compareTo(min) < 0 ||
            val.compareTo(max) > 0 ) {
            count++;
        }
    }
    return count;
}
```