CSM Berkeley 61B, Spring 2015: Week 2 Solutions

1. What's wrong with this picture?

File: Swap.java

```
class Swap {
    public static void swap(int a, int b) {
        int temp = b;
        b = a;
        a = temp;
    }

    public static void main(String[] args) {
        int x = 2;
        int y = 5;
        System.out.println("x: " + x + ", " + "y: " + y);
        swap(x, y);
        System.out.println("x: " + x + ", " + "y: " + y);
    }
}
```

a, b, and temp are local variables that exist only inside swap. The code inside swap only reassigns these names, which will not affect x or y. Drawing an environment diagram might help here.

In addition, since int is a primitive data type, a complete copy of the variable is made when it is reassigned, so x and a store two completely independent copies of the data 2.

2. What's different here?

File: Point.java

```
class Point {
    int x;
    int y;
    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```

File: Swap2.java

```
class Swap2 {
    public static void swap(Point p) {
        int temp = p.x;
        p.x = p.y;
        p.y = temp;
    }

    public static void main(String[] args) {
        Point x = new Point(2, 5);
        System.out.println("x: " + x.x + ", " + "y: " + x.y);
        swap(x);
        System.out.println("x: " + x.x + ", " + "y: " + x.y);
    }
}
```

Here, `Point` is an object and not a primitive data type, so assignment (the = operator) will not create more copies of a `Point`, but only new *references* or *pointers* to an existing `Point`. (Only the constructor, `Point()`, will create new `Points`.)

Here, `x` in `main` and `p` in `swap` are references to the same object, so changing `p.x` in `swap` will change `x.x` in `main`.

3. Write `skip`

`skip` takes in an IntList and destructively removes every other IntList node, starting at the *second* node. (So for example, 1, 2, 3, becomes 1, 3.)

Do NOT use recursion.

File: IntList.java

```
public class IntList {
    public int head;
    public IntList tail;

    public IntList(int head0, IntList tail0) {
        head = head0; tail = tail0;
    }

    public static void skip (IntList L) {

        while (L != null && L.tail != null) {
            L.tail = L.tail.tail;
            L = L.tail;
        }


    }

}
```

4. Extra Practice: What does `mystery` do?

Hint: draw box and pointers.

File: IntList2.java

```
public class IntList2 {
    public int head;
    public IntList2 tail;

    public IntList2(int head0, IntList2 tail0) {
        head = head0; tail = tail0;
    }

    public static IntList2 mystery(IntList2 L) {
        if (L == null || L.tail == null) {
            return L;
        } else {
            IntList2 x = mystery(L.tail);
            L.tail.tail = L;
            L.tail = null;
            return x;
        }
    }
```

`mystery` reverses the list.

```
    public String toString() {
        String result = "";
        IntList2 y = this;
        while (y != null) {
            result = result + y.head + " ";
            y = y.tail;
        }
        return result;
    }

    public static void main(String[] args) {
        IntList2 x = new IntList2(2, new IntList2(3, new IntList2(4, new IntList2(5, null))));
        System.out.println(x);
        IntList2 y = mystery(x);
        System.out.println(y);
    }
}
```