1 Pass by what?

Consider the following code block:

```
1  public class Pokemon {
2      public String name;
3      public int level;
4
5      public Pokemon(String name, int level) {
6          this.name = name;
7          this.level = level;
8      }
9
10     public static void main(String[] args) {
11         Pokemon p = new Pokemon("Pikachu", 17);
12         int level = 100;
13         change(p, level);
14         System.out.println("Name: " + p.name + ", Level: " + p.level);
15     }
16
17     public static void change(Pokemon poke, int level) {
18         poke.level = level;
19         level = 50;
20         poke = new Pokemon("Gengar", 1);
21     }
22 }
```

a) What will be printed out by the code above?

b) Draw a box and pointer diagram to illustrate what happened.

c) On line 19, we set level equal to 50. What level do we mean? An instance variable of the Pokemon class? The local variable containing the parameter to the change method? The local variable in the main method? Or something else?

2 Static Methods and Variables

```
1  public class Cat {
2     public String name;
3     public static String noise;
4
5     public Cat(String name, String noise) {
6         this.name = name;
7         this.noise = noise;
8     }
9
10    public void play() {
11       System.out.println(noise + " I'm " + name + " the cat!");
12    }
13
14    public static void anger() {
15       noise = noise.toUpperCase();
16    }
17    public static void calm() {
18       noise = noise.toLowerCase();
19    }
20 }
```

Write what will happen after each call of play() in the following method.

```
1     public static void main(String[] args) {
2         Cat a = new Cat("Cream", "Meow!");
3         Cat b = new Cat("Tubbs", "Nyan!");
4         a.play();
5         b.play();
6         Cat.anger();
7         a.calm();
8         a.play();
9         b.play();
10    }
```

3 Practice with Linked Lists

Draw a box and pointer diagram to represent the StringLists after each statement. A StringList is similar to an IntList. It has two instance variables, String head and StringList tail.

```
1         StringList L = new StringList("eat", null);
2         L = new StringList("shouldn't", L);
3         L = new StringList("you", L);
4         L = new StringList("sometimes", L);
5         StringList M = L.tail;
6         StringList R = new StringList("many", null);
7         R = new StringList("potatoes", R);
8         R.tail.tail = R;
9         M.tail.tail.tail = R.tail;
10        L.tail.tail = L.tail.tail.tail;
11        L = M.tail;
```

4 Bonus Problems: Squaring a List

Write the following methods to destructively and non-destructively square a linked list. Destructively squaring means modifying the items in the IntList passed as an argument. Non-destructively means all values in the original list are unaffected by the function.

```
/** Destructively squares each element of the given IntList L.
 * Don't use 'new'; modify the original IntList.
 * Should be written iteratively. */
    public static IntList SquareDestructive(IntList L) {









    }

/** Non-destructively squares each element of the given IntList L.
 * Don't modify the given IntList.
 * Should be written recursively*/
    public static IntList SquareNonDestructive(IntList L) {









}
```

Even more bonus problems: Write `SquareDestructive` recursively. Write `SquareNonDestructive` iteratively.