

1 Pass-by-What?

```
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 }
```

1.1 (a) What would Java display?

(b) Draw the box-and-pointer diagram after Java evaluates the `main` method.

(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? 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 }
```

2.1 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

3.1 Draw the box-and-pointer diagram that results from running the following code. A `StringList` is similar to an `IntList`. It has two instance variables, `first` and `rest`.

```
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.rest;
6 StringList R = new StringList("many", null);
7 R = new StringList("potatoes", R);
8 R.rest.rest = R;
9 M.rest.rest.rest = R.rest;
10 L.rest.rest = L.rest.rest.rest;
11 L = M.rest;
```

4 Squaring a List *Extra*

- 4.1 Implement `square` and `squareDestructive` which are static methods that both take in an `IntList L` and return an `IntList` with its integer values all squared. `square` does this non-destructively with recursion by creating new `IntLists` while `squareDestructive` uses a recursive approach to change the instance variables of the input `IntList L`.

```
1 public static IntList square(IntList L) {
```

```
1 public static IntList squareDestructive(IntList L) {
```

- 4.2 *Extra:* Now, implement `square` iteratively, and `squareDestructive` recursively.