Pass-by-What?

(a) Draw the box-and-pointer diagram after Java evaluates the `main` method. What would Java print?

(b) 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

```java
public class Cat {
    public String name;
    public static String noise;

    public Cat(String name, String noise) {
        this.name = name;
        this.noise = noise;
    }

    public void play() {
        System.out.println(noise + " I'm " + name + " the cat!");
    }

    public void nickname(String newName) {
        name = newName;
    }

    public static void anger() {
        noise = noise.toUpperCase();
    }

    public static void calm() {
        noise = noise.toLowerCase();
    }
}
```

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

```java
public static void main(String[] args) {
    Cat a = new Cat("Cream", "Meow!");
    Cat b = new Cat("Tubbs", "Nyan!");
    a.play();
    b.play();
    Cat.anger();
    a.calm();
    a.play();
    b.play();
    a.nickname("Kitty");
    a.play();
    b.play();
}
```

(b) If we were to add `Cat.nickname("KitKat")` to the end of our main function, what would happen?
3 Practice with Linked Lists

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.

```java
StringList L = new StringList("eat", null);
L = new StringList("shouldn't", L);
L = new StringList("you", L);
L = new StringList("sometimes", L);
StringList M = L.rest;
StringList R = new StringList("many", null);
R = new StringList("potatoes", R);
R.rest.rest = R;
M.rest.rest.rest = R.rest;
L.rest.rest = L.rest.rest.rest;
L = M.rest;
```
4 Squaring a List

Extra

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

\begin{verbatim}
public static IntList square(IntList L) {
}

public static IntList squareDestructive(IntList L) {
}
\end{verbatim}

Extra: Now, implement \texttt{square} iteratively, and \texttt{squareDestructive} recursively.