CS 61B
Spring 2018

Scope, Pass-by-Value, Static

Discussion 2: January 23, 2018

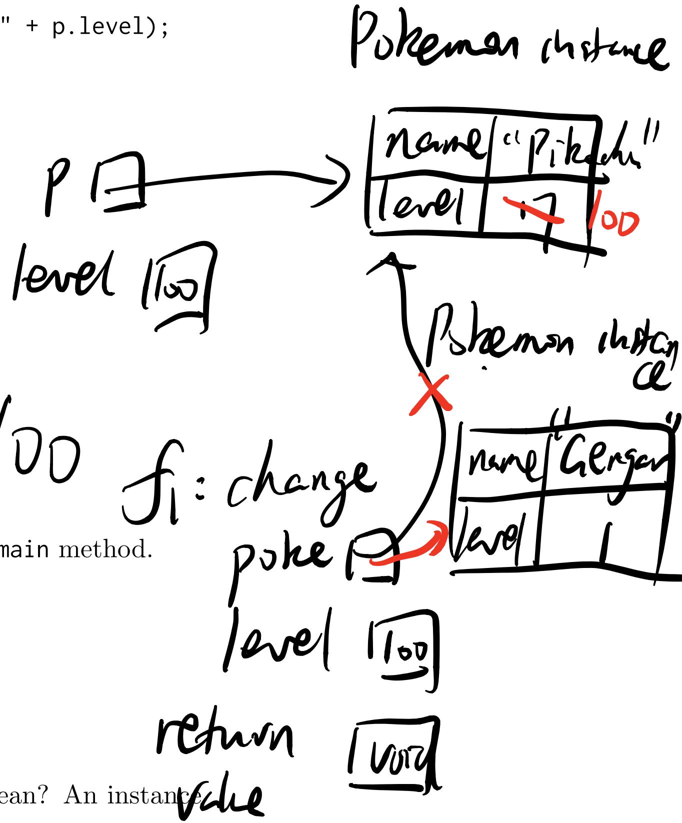
1 Pass-by-What?

```
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?

Name: Pikachu, Level: 100

(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?

✓

an local variable containing the parameter to the change method.

2 Static Methods and Variables

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

Instance variable

ノイズ

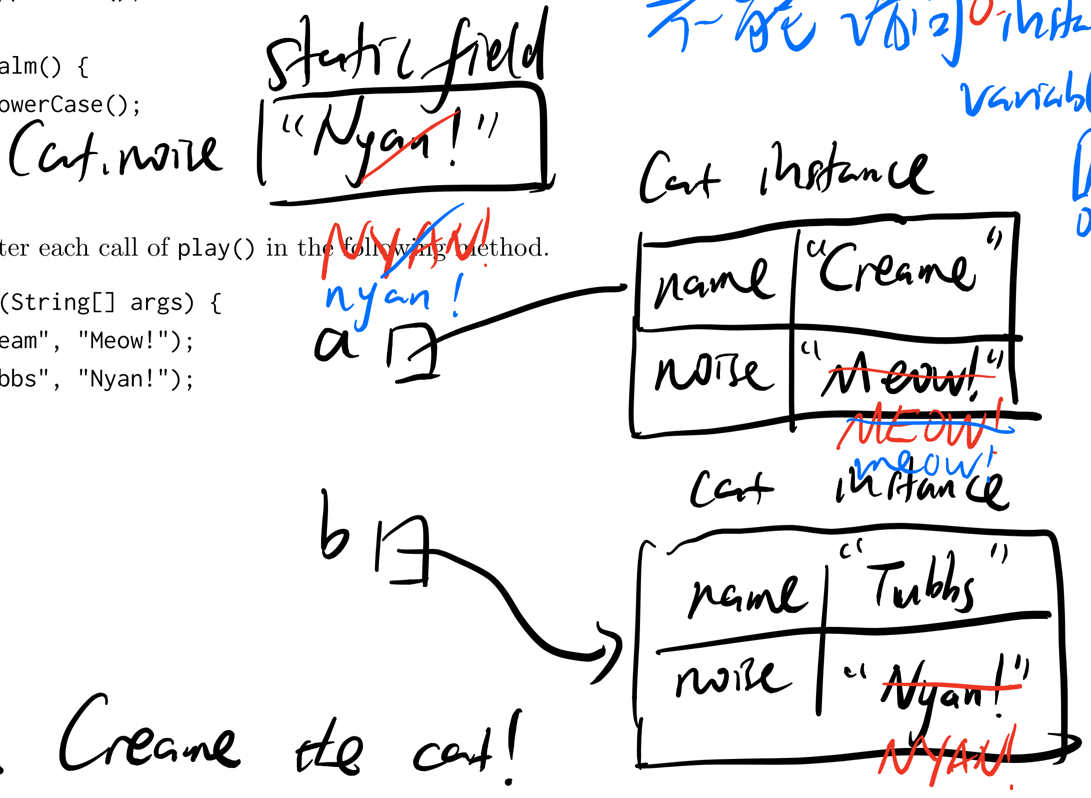
class Variable

noise is declared to be a static variable, which means ~~that there~~ is only one noise variable for the entire Cat class.

static method can only (access/refrence) modify static variables!

不能访问instance variable!

0,



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

```
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();
}
```

Nyan!

4: ~~Meow!~~ I'm Creame the cat!

5: Nyan! I'm Tubbs the cat!

6.

7.

nyan!

8. ~~Meow!~~ I'm Creme the cat!

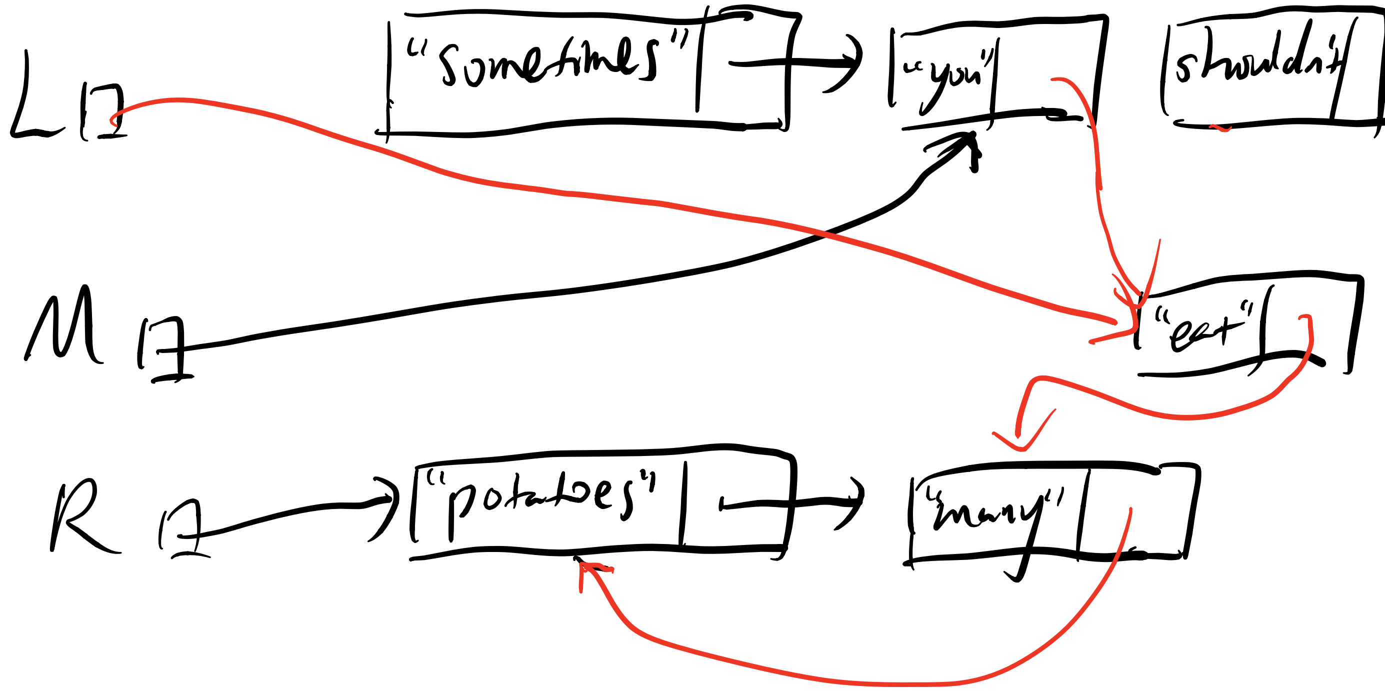
9. ~~NYAN!~~ I'm Tubbs the cat!

nyan!

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`.

```
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*

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`.

```
public static IntList square(IntList L) {
        if (L == null) {
            return L;
        }
        return new IntList(L.first * L.first, square(L.rest));
}.
```

```
public static IntList squareDestructive(IntList L) {
        if (L==null) {
            return L;
        }
        L.first = L.first * L.first;
        squareDestructive(L.rest);
```

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

```
        return L;
}.
```