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.

see Java Visualizer

(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?

it is just a local variable in the change method. iIt does not have any effect on the other variables of the same name in the Pokeman class or the main method.

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 }
```

note that noise is a class attribute, which is the same for every cat instance

when instantiating a, Cat.noise becomes Meow, then Nyan when instantiating b

both static methods will have effect on the class attribute noise

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 }
```

MeowNyan! I'm Cream the cat!

Nyan! I'm Tubbs the cat!

meownyan! I'm Cream the cat!

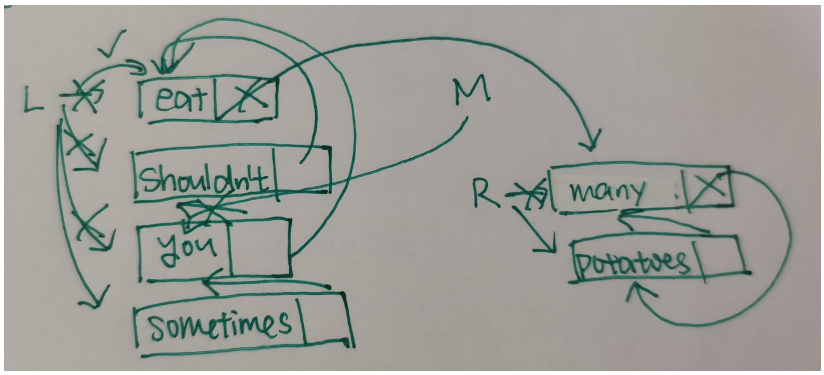
NYANnyan! I'm Tubbs the cat!

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;
```

correct



4 Squaring a List *Extra*

4.1 Implement `square` and `squareMutative` 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-mutatively with recursion by creating new `IntLists` while `squareMutative` uses a recursive approach to change the instance variables of the input `IntList L`.

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

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

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

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

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

```
public static IntList square(IntList L) {
    /*
    IntList ans = new IntList(L.first * L.first, null);
    if (L.rest != null) {
        ans.rest = square(L.rest);
    }
    return ans;
    */
    IntList ans = new IntList(L.first * L.first, null);
    IntList ptr1 = ans;
    IntList ptr2 = L.rest;
    while (ptr2 != null) {
        ptr1.rest = new IntList(ptr2.first * ptr2.first, null);
        ptr1 = ptr1.rest;
        ptr2 = ptr2.rest;
    }
    return ans;
}
```