CIS 256 — Scope, Pass-by-Value, Static

Fall 2019

1 Pass-by-What?

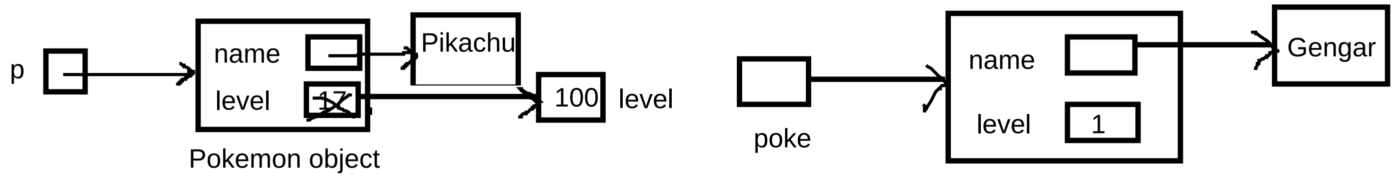
```
1  public class Pokemon {
2      public String name;
3      public int level;
4
5      public Pokemon(String name, int level) {
6          this.name = name;
7          this.level = level;
8      }
9
10     public static void main(String[] args) {
11         Pokemon p = new Pokemon("Pikachu", 17);
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?

java will 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?

I think the level is the local variable containing the parameter to the method

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

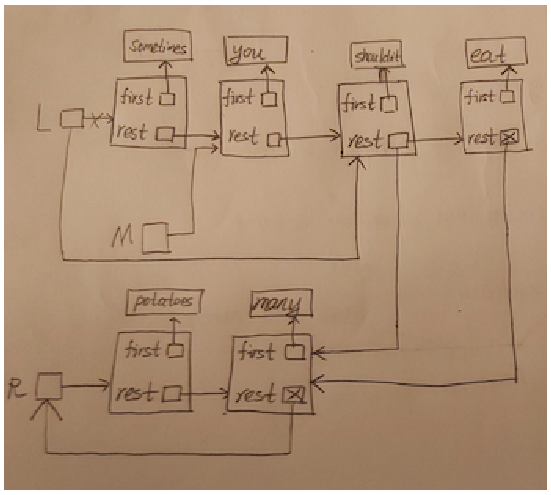
2.1 Write what will happen after each call of `play()` in the following method.

```java
public static void main(String[] args) {
    Cat a = new Cat("Cream", "Meow!");
    Cat b = new Cat("Tubbs", "Nyan!");
    a.play();   Meow! I'm Cream the cat!
    b.play();   Nyan! I'm Tubbs the cat!
    Cat.anger(); Nyan! -> NYAN!
    a.calm(); NYAN! -> nyan!
    a.play(); nyan! I'm Cream the cat!
    b.play(); nyan! 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`.

```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;
```

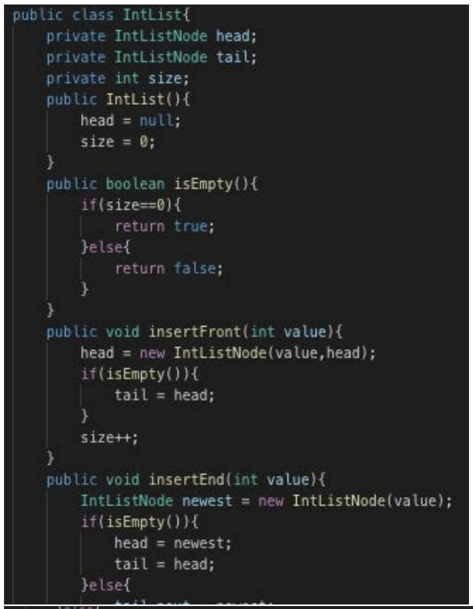


4 Squaring a List *Extra*

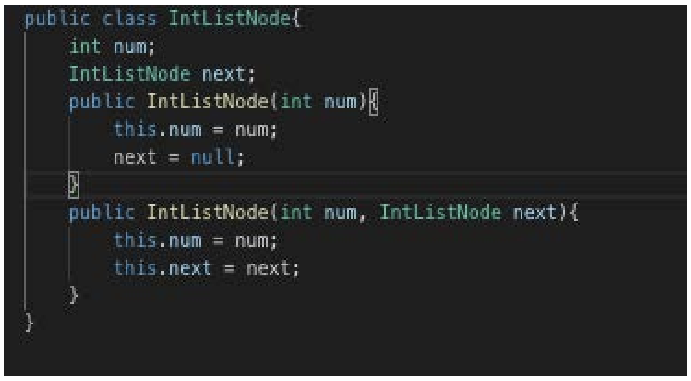
4.1 Implement `square` and `squareDestructive` which are static methods that both take in an `IntList L` and return an `IntList` with its integer values all squared. `square` does this non-destructively with recursion by creating new `IntLists` while `squareDestructive` uses a recursive approach to change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

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

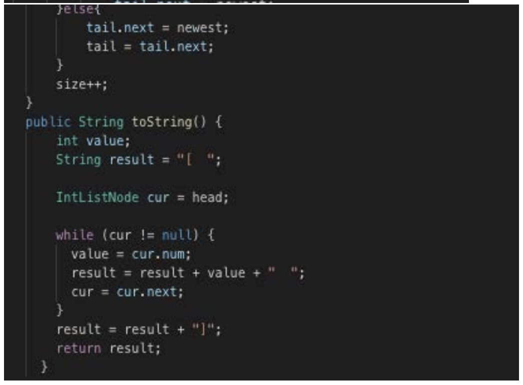
```
//iteratively
  public static IntList square(IntList L){ // return an IntList after squaring
    IntListNode cur = L.head;
    IntList squaredList = new IntList();
    while(cur!=null){
      squaredList.insertEnd(cur.num*cur.num);
      cur = cur.next;
    }
    return squaredList;
  }

//recursively
  public static IntListNode square(IntListNode L){
      if(L==null){
        return null;
      }else{
        return new IntListNode(L.num*L.num, square(L.next));
      }
    }
```



```
public class IntList{
    private IntListNode head;
    private IntListNode tail;
    private int size;
    public IntList(){
        head = null;
        size = 0;
    }
    public boolean isEmpty(){
        if(size==0){
            return true;
        }else{
            return false;
        }
    }
    public void insertFront(int value){
        head = new IntListNode(value,head);
        if(isEmpty()){
            tail = head;
        }
        size++;
    }
    public void insertEnd(int value){
        IntListNode newest = new IntListNode(value);
        if(isEmpty()){
            head = newest;
            tail = head;
        }else{
```



```
public class IntListNode{
    int num;
    IntListNode next;
    public IntListNode(int num){
        this.num = num;
        next = null;
    }
    public IntListNode(int num, IntListNode next){
        this.num = num;
        this.next = next;
    }
}
```

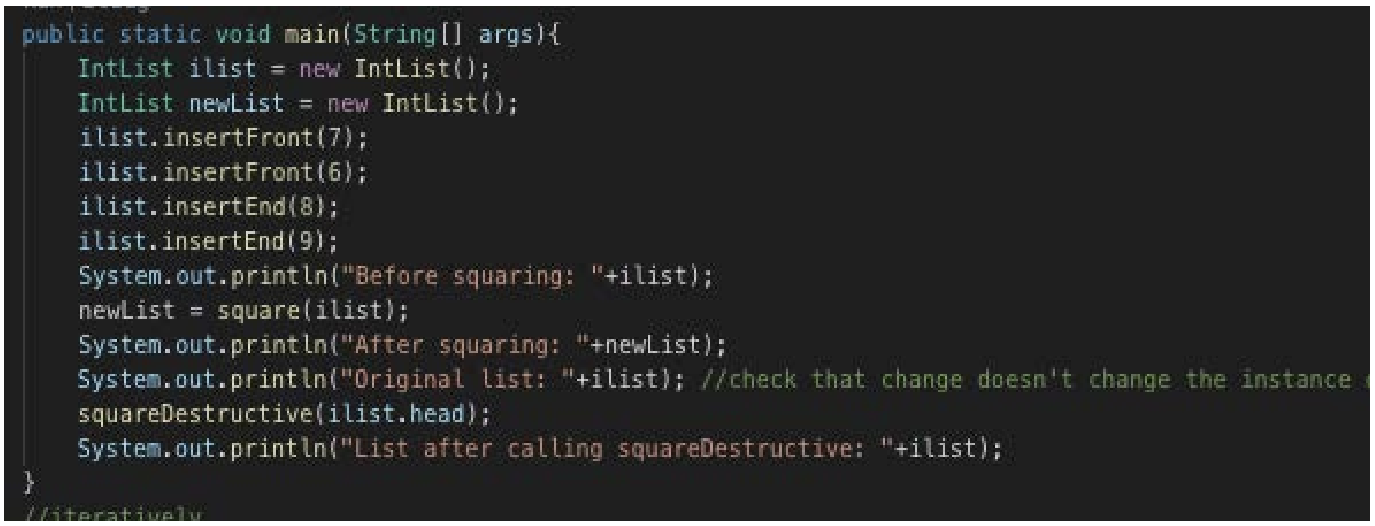


```
        }else{
            tail.next = newest;
            tail = tail.next;
        }
        size++;
    }
    public String toString() {
        int value;
        String result = "[  ";

        IntListNode cur = head;

        while (cur != null) {
          value = cur.num;
          result = result + value + "  ";
          cur = cur.next;
        }
        result = result + "]";
        return result;
      }
```

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

        public static IntListNode squareDestructive(IntListNode L){
            if(L==null){
                return null;
            }else{
                L.num = L.num*L.num;
                return squareDestructive(L.next);
            }
        }
```

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

implementation



```
public static void main(String[] args){
    IntList ilist = new IntList();
    IntList newList = new IntList();
    ilist.insertFront(7);
    ilist.insertFront(6);
    ilist.insertEnd(8);
    ilist.insertEnd(9);
    System.out.println("Before squaring: "+ilist);
    newList = square(ilist);
    System.out.println("After squaring: "+newList);
    System.out.println("Original list: "+ilist); //check that change doesn't change the instance
    squareDestructive(ilist.head);
    System.out.println("List after calling squareDestructive: "+ilist);
}
//iteratively
```