CS61B Discussion 2

Classes, Pass-by-Value, Static

Administrivia

- Lab 1 and Lab 2 due this Friday 2/1 at midnight.
- Proj0 NBody also due Friday 2/1 at midnight.
  - If you haven't started please start ASAP!
- One-on-one tutoring will be made available the week of **Feb. 3rd!** We'll announce details soon.
- We're sending out weekly announcements! Check out this week's @346
- Office hours, LOST and exam prep sections start this week! See Piazza post @346 for more info.

What are we doing today?

- Review
  - Class definition, primitives and reference, box and pointer, static, intlists
- Questions
  - Will get through 1, 2, 3
  - Likely won’t get to 4

Review: Defining classes

```
// class definition
public class IntList {

     // instance variables
     int head;
     IntList tail;

     // constructors, which will create IntList objects
     public IntList() {...}
     public IntList(int value, IntList tail) {...}

     // you can create methods that use the instance variables!
     public void insert(int val) {...}
     public void removeLast() {...}
}
```

Review: Primitives and Reference

Primitives:

- byte, short, long, int, double, float, char, boolean

References:

- Everything else!
- Objects, arrays, Strings, etc.

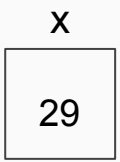
Golden Rule of Equals AKA pass-by-value

- When you do “a = b”, you always copy the **bits** of b into a.
- **For primitives:** Directly copy the value inside b to a.
- **For Reference types:** Copy the memory address of b into a.
  - Also can say “copies the pointer in b to a”

Review: Box and Pointer Diagrams

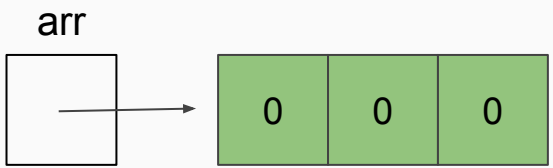
- Box and Pointers are the 61B equivalent of 61A's environment diagram
- They are crucial to helping us keep track of what goes on in a program: **use them!**

```
int x = 29;
```



Primitives go in the box

```
int[] arr = new int[3];
```



Reference types are referenced via pointers

We didn't specify what integers are going in the array, so Java initializes it with the default value 0

Review: Static

```
Public class Dog {
      public int legs;
      public static String name = “Doge”;
      public static void bark() {...}

      public static void main(String[] args) {
            Dog fido = new Dog();
      }
}
```

- Static variables are shared by all instances of the class and should be accessed via dot notation with the **class**, not the instance.
  - `Dog.name` instead of `fido.name`
- Static methods are methods that do not require an instance to call it, and thus do not access any instance variables.
  - `Dog.bark();`
- You **can** call static methods on an instance, but what will happen is Java will actually look for the static type of the object and run the method from that class.

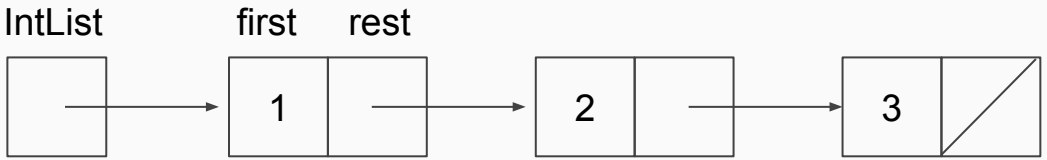
```
Dog g = new Dog();
g.bark();
```

=

```
Dog g = new Dog();
Dog.bark();
```

Review: IntLists

- IntLists are the equivalent of 61A's "Linked List" with one restriction: it can only hold integers.
- Each IntList consists of 2 elements:
  - an integer `first`
  - an IntList `rest`



1.) Pass by What?

Consider the following code block:

```
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) Pass by What?

- What would Java display?
  - Name: Pikachu, Level: 100
- What does the Box and Pointer look like after Java evaluates the main method?
  - Java Visualizer
- 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 the local variable in the change method and does not have any effect on the other variables of the same name in the Pokemon class or the main 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();
    }
}
```

```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();
}
```

Java Visualizer

Detailed explanation is also on the discussion solution!

3.) Practice with Linked Lists

```
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;
```

Java Visualizer

4.) Extra: Squaring a List

Components of recursion:

1. Base case
2. Recursive call
3. Assuming the recursive call works, how can we use it?

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

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

4.) Extra: Squaring a List

Components of recursion:

1. Base case
2. Recursive call
3. Assuming the recursive call works, how can we use it?

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

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

4.) Extra: Squaring a List

Components of recursion:

1. Base case
2. Recursive call
3. Assuming the recursive call works, how can we use it?

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

    IntList B = L;
    while (B != null) {
        B.first *= B.first;
        B = B.rest
    }
    return L;
}
```