Scope, Pass by Value, and Static Types

Discussion 02

Announcements

Week 2

- ❏ Live lectures are now Q&A Sessions
- ❏ Weekly Survey due every Sunday
- ❏ Discussion & Lab attendance is optional
- ❏ Lab partnerships are completely optional
- ❏ Exam Prep sections start next week - notify us in the google form if you're interested
- ❏ OH start this week on Discord! Read up on what that means before you show up
- ❏ If you need DSP accommodations for exams fill our the form on Ed

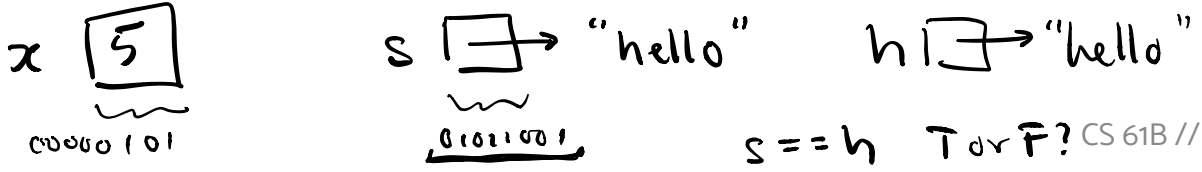
Content Review

Primitive vs. Reference Types

Primitive Types are represented by a certain number of bytes stored at the location of the variable in memory.
Examples: Bytes, Short, Int, Long, Float, Double, Boolean, and Char (that's it!)

Reference Types are represented by a memory address stored at the location of the variable which points to where the full object is.
Examples: Strings, Arrays, Linked Lists, Dogs, etc.

== compares the information at the location of the variable - so it only works for primitive types unless you are trying to compare the memory address of two object, you want to use **.equals()**

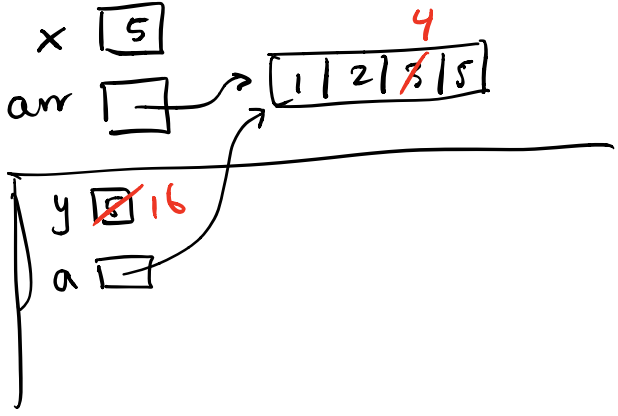


Pass by Reference

```
...
int x = 5;
int[] arr = int[]{1, 2, 3, 5};
doSomething(x, arr);
...

public void doSomething(int y, int[] a) {
    y = 16;
    a[2] = 4;
}
```

After running the first code block, x = 5 and arr = [1, 2, 4, 5].



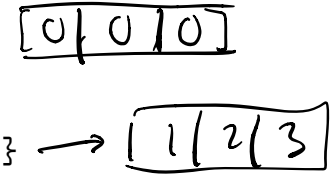
Arrays

Arrays are lists of items that can be indexed into using bracket notation. They are zero-indexed, so the first item is at index 0.

Example: `arr[2]`

Instantiation:

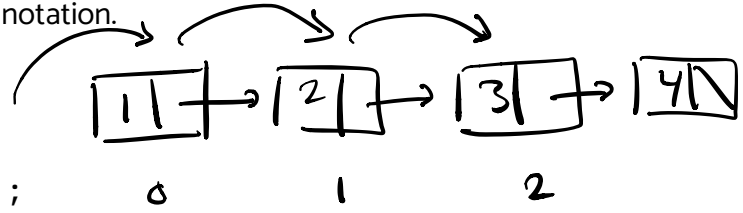
```
int[] x = new int[3]
int[] x = new int[]{1, 2, 3}
```



Once an array is created, its size is **immutable**, or unchangeable.

Linked Lists

Linked Lists are modular lists that are made up of nodes that each contain a value and a pointer to the next node.
To access values in a Linked List, you must use dot notation.
Example: `intList.get(2)`



Instantiation:

```
IntList intList = new IntList();
```

They can be extended or shortened by changing the pointers of its nodes (unlike Arrays).

They can't be indexed directly into like an array, instead the computer has to iterate through all of the nodes up to that point and follow their next pointers.

Worksheet

CS 61B Fall 2020

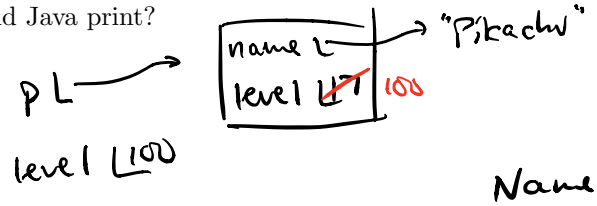
Scope, Pass-by-Value, Static

Discussion 2: August 31, 2020

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 }
```

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



Name: Pikachu, Level: 100

(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?

parameter inside change 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 void nickname(String newName) {
15       name = newName;
16    }
17
18    public static void anger() {
19       noise = noise.toUpperCase();
20    }
21
22    public static void calm() {
23       noise = noise.toLowerCase();
24    }
25
26 }
```

← instance variable

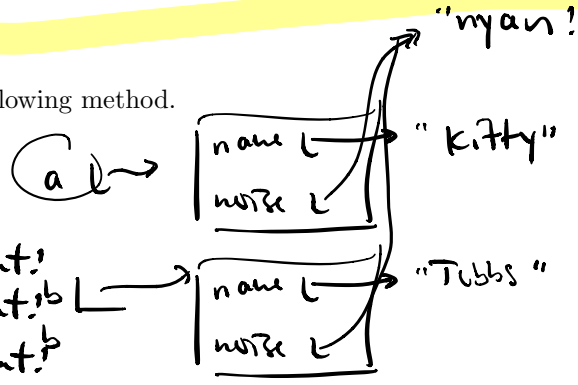
← class variable

1. static methods can't use instance variables
2. inst. var/methods are ONLY accessible through other instances of obj.
3. static vars/instances belong to whole class (also instances)

(a) 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     a.nickname("Kitty");
11     a.play();
12     b.play()
13 }
```

Nyan! I'm Cream the cat!
Nyan! I'm Tubbs the cat!
NYAN! I'm Cream the cat!
nyan! I'm Tubbs the cat!
nyan! I'm Kitty the cat!
nyan! I'm Tubbs the cat!



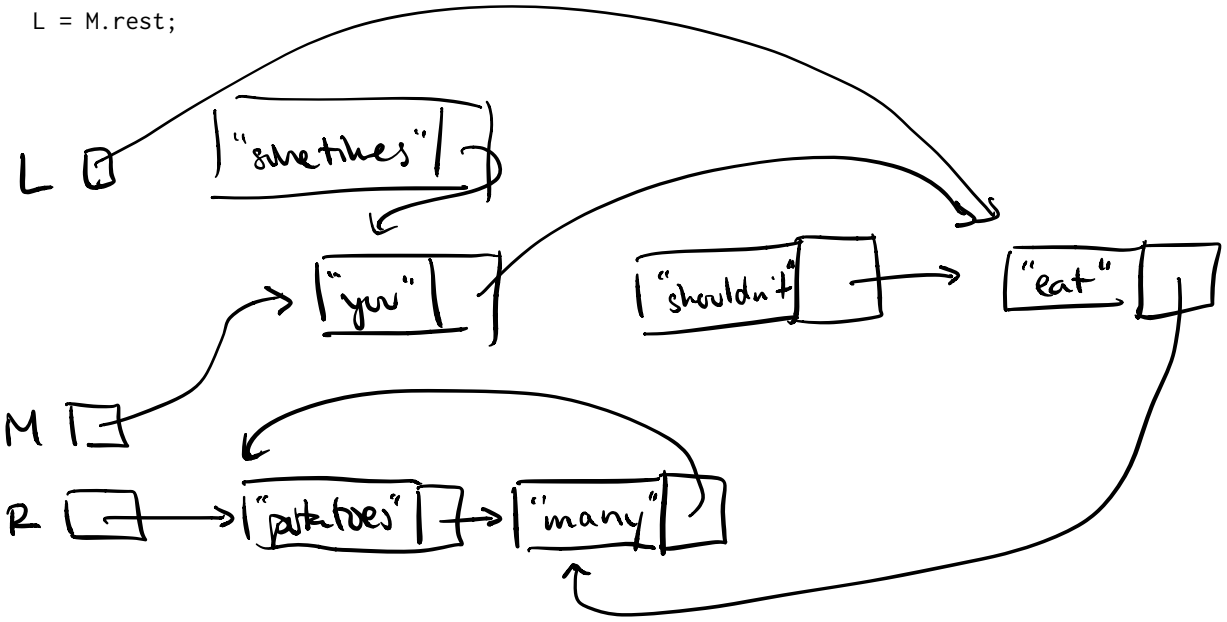
(b) If we were to add `Cat.nickname("KitKat")` to the end of our main function, what would happen?

nickname is instance method!
Cannot be called from static context!

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`.

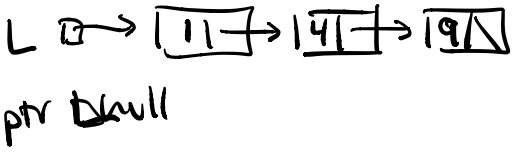
```
1  StringList L = new StringList("eat", null);
2  L = new StringList("shouldn't", L);
3  L = new StringList("you", L);
4  L = new StringList("sometimes", L);
5  StringList M = L.rest;
6  StringList R = new StringList("many", null);
7  R = new StringList("potatoes", R);
8  R.rest.rest = R;
9  M.rest.rest.rest = R.rest;
10 L.rest.rest = L.rest.rest.rest;
11 L = M.rest;
```



4 Squaring a List *Extra*

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 an iterative approach to change the instance variables of the input `IntList L`.

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




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

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