Classes: references

Hey, don't I know you?

Don't you know that pointing is rude?
Highlights:

```java
Pet dog = new Pet();
Pet cat = dog;
```

What classes store (references)

```java
Pet p = new Pet();
p(t)(p);
```

Passing classes into methods
Announcements

Jina added an office hour after class (Keller 4-240)

(Reminder, my office hours are at 5-6pm)

Midterm next week:
- Covers up to and including loops (HW3/lab4)
- Open book/note (no compiler/netbeans)
- Will post a sample midterm
public vs. private

Using **public** before a variable/function declaration ensures anyone can use it.

Using **private** ensures only types of that class can use it (but not just the original class).

**TL;DR** private is not per object, but per class.

```java
Dog d1 = new Dog();
Dog d2 = new Dog();
d1.change(d2);  // d1 can change privates of d2
```

(See: Dog.java and Cat.java)
Classes vs. primitive types

When you say:  `int x;` ... you create a box for x (that holds an int)

When you say:  `Point y;` ... you create a box that references another box

This reference originally points nowhere, you need “new” and a constructor to make an instance of the class (a box for data)
Every variable has to be stored in the computer's memory.

Normally, we think of memory as a big long line with variables taking up more or less space on the line.

This is true for primitive types (int, double), but not for classes...
References

(See: InfiniteMemory.java)
When memory does not actually hold the value of an object, but instead holds information about the actual location...

... this is called a reference
References

Another example: a phone number is a reference to a person/phone

A phone number simply connects you to another person

If two people have the same phone number (reference), they will call the same person (instance of an object)
If you use `Class1 == Class2`, this will compare references!

For classes, you should use `.equals()` instead (you have to make it)

`null` is a reference to nothing (the abyss)

(See: ClassLink.java)
As the “value” of a class is the reference (phone number) and methods copy the value...

If you pass a method into a class, it can be changed back in the original method (main)

This is because they both “call” the same instance of the object
(See: ChangingClass.java)
Using classes in methods

How can you get around this?
Using classes in methods

How can you get around this? Do what primitive types do: pass a copy

To do that we need to make a copy, and thus a new object with identical values

A way to do this is to make a copy constructor (See: CopyConstructor.java)
Wrapper classes

Class versions of primitive types (int, double, char, etc.) exist.

These Class versions are nice because they include useful methods.

For example
Double x = new Double(2);
int y = Integer.parseInt("-412");
Q: It seems you should have information about yourself, but how do you access that?

A: Inside every class, there is a `this` variable, that references yourself (i.e. `this` = your own phone number).
this

this is a command for the class object

The this command is useful when you need to be explicit about which method or object to access (also when linking)

(See: ThisCommand.java)