Classes: static & instances

STATIC ELECTRICITY

"Yeah, really funny... rub me on the carpet and then put me in the shipping box... You will pay for this!"
Highlights:

```java
public void justAnotherName() {
}
```

Static or not?

```java
public static void main(String[] args) {
```
class

Every time you do not make a primitive type (int, double, boolean, etc.) or String, you must use **new** command

Basic format (syntax):
[class] [identifier] = **new** [class]();

Example:
Partner p1 = **new** Partner();
Scanner in = **new** Scanner(System.in);
class

Suppose we wanted to keep scores of lab0, lab1, lab2, lab3 and hw1 for 5 students

Without classes how would I have to do this?

With classes how would I do this?
(See: Classes.java)
classes are outlines/blue prints of an organization structure

Thus when you create a variable of your class's type, you create an instance
Consider the word “human”...
Can think of it as a type (label/blueprint):
English: James is a human.
Code: Human james;

Some properties are shared by all humans, such as the fact that we are mammals

Other properties are specific to each human, my name is (somewhat) unique to me
static

The static modifier makes only one for the entire class (shared)

If you do not have static, it can be different for each instance of the class

This causes slightly different effects in methods and variables

(See Shared.java)
If a method is **static**, then you can access this method without creating a variable for that class.

For example: Math.pow()

(See: RunStaticMethod and StaticMethod)
static methods

Benefits:
- Can be accessed without using a variable

Annoyances:
- Cannot modify non-static class variables
- Cannot use non-static methods (in case they might modify the class)
static variables

When you make a static variable, then all all classes see the same variable

If one class changes the static variable, then that variable is changed everywhere

(See: Profile.java)
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.
When “passing” arguments to methods, we always go in and just give them the value (and not tell them where we are storing it)

This means that primitive types (int, double) cannot have their values changed inside methods, and class variables cannot suddenly reference some other object
If you use Class1 == Class2, this will compare references!

null is a reference to nothing

(See: ClassLink.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 pointer, that points to yourself.
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)