Classes: references part 2
Ch. 5.1-5.3

“pointers” are “references”

memory locations
Highlights:

```java
Dog x = new Dog("Ziggy");
Dog y = new Dog("Brownie");
if(x.equals(y))
```

Using `equals()`

```java
Sheep dolly = new Sheep("Dolly");
Sheep clone = new Sheep(dolly);
```

copy constructor
If a variable references `null`, then you will crash your program if you use the reference.

For example:

```java
String x = null;
int y = x.length();
```

This will give a “Null Pointer Exception”, (you should be able to understand errors now)
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** 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)
When using classes, you should use `.equals()` instead of `==` (more on this later)

However, you have to make this method, as the default one doesn't work (same as `==`)

```java
public boolean equals(EqualsCompare o) {
    if (o.x == x && o.y==y) {
        return true;
    }
    return false;
}
```

(See: EqualsCompare.java)
If data is private, then it can only be changed through your methods, right?

Wrong

Classes are stored as references...

If you give out its reference, then it can be changed as the user desires (See: Next.Java)
Using classes in methods

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
Using classes in methods

If we just use an “=” , it will simply copy the reference (phone number) and not the whole class (and all the values)

FakeClass x = new FakeClass();
FakeClass y;
y=x; // variables simply have same reference

This will not help, since we still have only one box, so they are sharing
In fact, we want another box so we need to create a new object (thus need a constructor).

We can build a special one for copying:

```java
public class FakeClass{
    public FakeClass(FakeClass toCopy)
    {
        // copy all in here! toCopy.values = values
    }

(See: CopyConstructor.java)