a first look at

Classes and Objects

continued next week
Classes and Objects

Java is an object-oriented language
but we haven’t been using OO features

What’s it mean?

(Your text has a good explanation ... read it!)
Another way to arrange things

Arrays give us rows of boxes ...

But they are all the same. Maybe I don’t want a row. I want different kinds of boxes, with names, in a bigger box.

And it should be black. Definitely black.
Object

There’s stuff inside, but it’s private.

Just tell me what to do with it.
There’s a tic-tac-toe board in there ...

But I don’t see how it’s represented. The object just provides some methods I can call, like

```
board.placeX(3,3)
```

and

```
board.isWinForX()
```
You’ve seen this already ...

Like “Scanner” objects

Scanner f = new Scanner( ... )

while (f.hasNext() ) {
    s = f.next()
}

f is an object
Scanner is its class

There’s a lot of stuff inside!
Classes and Objects

Objects are made from classes

E.g., scanner objects are made from class Scanner

If we wanted a TicTacToe board object
We’d write a Class TicTac
And then ...

TicTac board = new TicTac();
I can put (named) boxes in my box ... they’re private unless I make them public.
If I make two objects from the same class, each one has its own copies of the fields (like making two scanners from class Scanner)
Let’s design a Tic Tac Toe board class ...

```java
public class TicTac {

    TicTac board = new TicTac();

}
```
Encapsulating data and functions

A class can be a module that groups some data with related functions

- A convenient unit of reuse and change
- Often a reasonable brain-size chunk (if designed well)

But classes aren’t the only kind of module ... and that still doesn’t explain the wacky backward syntax object.method(arg, ...)

why not function(object, arg, ...)?
“Object-oriented” features (classes and objects) were developed in SmallTalk and spread to many other languages. Originally for simulation (Simula 67), then for modularity.
OO and Java

Java programs are collections of classes
Methods only.
No functions outside of classes.

Static means: I want to call it without an object!
(So, before we had objects, we used static a lot)
Object-oriented = Good?

Sometimes!

Often useful, not always. Helps with certain kinds of program evolution

Not the only way to modularize programs. Not always the best way

Widely used. You need to learn it, but you don’t always have to choose it.
Polymorphism in method dispatch

poly = many, morph = form
same method call can work for several
different kinds of data

Example: suppose we have a list of shapes,
including triangles, squares, and circles. We
want to add up their areas.
We don’t want this:

total = 0;
for (int i=0; i < shapelist.length; ++i) {
    Shape shape = shapelist[i];
    if (is_square(shape))
        { total += square_area(shape); }
    else if (is_triangle(shape))
        { total += tri_area(shape); }
    else if (is_circle(shape))
        { total += circ_area(shape); }
}

Imagine we have a lot of code like this, and then we need to add support for ellipses. Yuck.
Better:

```java
int total = 0;
for (int i = 0; i < shapelist.length; ++i) {
    Shape shape = shapelist[i];
    total += shape.area();
}
```

There is still an area function for squares, and an area function for triangles, and one for circles ... but each one is a method in the corresponding class.

If we need to add a class for ellipses, no change needed here!

We’ve localized some kinds of program changes, which is a GOOD THING (even worth putting up with wacky syntax)
objects.call().methods().backward().you_must()

but master, why?

What’s the point of classes and objects?
Why *backward* you must write?

total = 0;
for (int i=0; i < shapes.length; ++i) {
    total += shapes[i].area();
}

Use the pointers, Luke

*image source: Wookiepedia, screen shot from Star Wars Episode IV: A New Hope*
Summary

Classes and Objects provide a way to divide programs into modules
- Grouping data and related functions into meaningful, potentially reusable chunks

Polymorphic, dynamic dispatch in method calls helps localize changes
- Adding a new, related class is simpler than changing a lot of code to call the appropriate functions

A widely used approach
- Often useful, worth learning, but not always best