Module Introduction

- In this module, we will see how classes, objects, and interfaces fit into our account of information analysis and data design.
- We'll see how the functional and the object-oriented models are related.
- We'll learn how to apply the design recipe in an object-oriented setting.
Goals of this lesson

• Learn the basics about classes, objects, fields, and methods.

• Learn how these ideas are expressed in the Racket object system
What is an object?

- An object is another way of representing compound data, like a struct.
- Like a struct, it has **fields**.
- It has one built-in field, called **this**, which always refers to this object.
- Here are pictures of two simple objects:

```
x = 10
y = 20
r = 10
this = ●

h = 30
w = 15
color = "blue"
this = ●
```

We assume that you've seen some kind of object-oriented programming before, so we're just reviewing vocabulary here.

If you've really never used OOP before, go do some outside reading before continuing.
How do you compute with an object?

• Every object comes equipped with a set of procedures, called *methods*.
• Each method has a name.
• To invoke a method of an object, we send the object a message.
• For example, to invoke the area method of an object *obj1*, we write

  (send obj1 area)
Classes

• Every object has a class.
• The class specifies the fields of the object. 
  – so it's like a define-struct.
• The class contains the methods of that object.
• In a typical design, we are likely to have many objects of the same class.
• To create an object, we say
  \[(\text{new } C)\]
  where \(C\) is the name of the new object's class.

You also need to give new initial values for the fields, but we’ll get to that later.
Every object knows its class (1)

Here are two objects of the same class. In the class definition, the init-field declaration specifies that each object of this class has 3 fields, named x, y, and r. The class definition also defines two methods, named foo and bar, that are applicable to any object of this class.

These objects also have a this field, but we don't show it unless we need to.
Every object knows its class (2)

```
(class* object% ()
  (init-field x y r)
  (define/public (foo) (+ x y))
  (define/public (bar n) (+ r n))
  ...)
```

The variables in the method declarations refer to the fields in the object. So:

- (send obj1 foo) returns 30
- (send obj2 foo) returns 50
Every object knows its class (3)

(class* object% ()
  (init-field x y r)
  (define/public (foo) (+ x y))
  (define/public (bar n) (+ r n))
  ...)

Methods can also take arguments, just like functions. So

(send obj1 bar 8) returns 18
(send obj2 bar 8) returns 13
Every object knows its class (4)

Methods are just Racket functions, so they can do anything a Racket function can do, including send messages to objects.

\((\text{send obj1 baz 20})\) returns \((+ 30 20) = 50\)

\((\text{send obj2 baz 20})\) returns \((+ 50 20) = 70\)
Every object knows its class (5)

Here's another object, obj3, of a different class (observe that the bar method is different). If we send a message to obj3, then obj3's methods will be invoked.
Every object knows its class (6)

```
(class* object% ()
  (init-field x y r)
  (define/public (foo) (+ x y))
  (define/public (bar n) (+ r n))
  (define/public (baz n) (+ (send this foo) n)))

(obj1)
  x = 10
  y = 20
  r = 10
  this = ...

(obj2)
  x = 15
  y = 35
  r = 5
  this = ...

(obj3)
  x = 15
  y = 35
  r = 5
  this = ...
```

So (send obj2 bar 8) = (+ 5 8) = 13
(send obj3 bar 8) = (- 5 8) = -3
Using The Racket Class System

• We will use full Racket (yay!)
• Write
  
  #lang racket
  
  at the beginning of each file
• And set the Language level to "Determine Language from Source"
First demonstration system: space-invaders.rkt

• A simple animated system using the universe module and the Racket object system

• Specifications:

• We have classes for
  – worlds
  – bombs
  – helicopters
Game Description

• When the system starts, the world contains just a helicopter
• the helicopter rises at a constant rate
• Press space to drop a new bomb
• Bombs fall at a constant rate
• Bombs are draggable
Goal

• We'll walk through the code of this system to illustrate the Racket object system.
Demonstration: space-invaders.rkt

• Demonstration and code walkthrough 10-1-space-invaders.rkt

• Demonstration:  
  http://youtu.be/hbcPu5B8q40 (0:48)

• Walkthrough:  
  – Part 1:  http://youtu.be/Pbc0ruZb33U (7:09)  
  – Part 2:  http://youtu.be/HSZGSj04LI0 (7:03)

As with other videos, these videos were recorded earlier, and may not represent our best current practice. In particular, they use Number instead of Integer.
Lesson Summary

• We’ve learned the basics about classes, objects, fields, and methods.
• We’ve seen how these ideas are expressed in the Racket object system