Reading:
- Mitchell, Ch. 11

Objectives
1. be able to describe in what ways SmallTalk can be considered a more ‘dynamic’ language (than C++, Java, or Scala)
2. be able to diagram the memory layouts SmallTalk uses for method and field lookup

Outline
1. history: Alan Kay’s Dynabook at Xerox PARC (portable computer with GUI)
2. object model
   (a) everything is an object (even integers, strings, boolean values, blocks of code)
   (b) all operators are messages [method calls] to objects
   (c) objects and classes are used to organize entire programs
   (d) information hiding: private instance variables, public methods
   (e) inheritance is code reuse unrelated to subtyping
3. flexibility and dynamicism
   (a) no static typing (like Lisp/Scheme, Python, Ruby, Groovy)
   (b) dynamic lookup of methods by ‘selectors’ [method name]
   (c) Dan Ingalls: “Can you define a new kind of integer, put your new integers into rectangles, ask the system to blacken a rectangle, and have everything work?”
   (d) subtyping is implicit and not enforced by the language (“duck typing”)
   (e) cost: performance, reliability
   (f) design patterns, test driven development, agile software development
4. runtime dispatch mechanisms
   (a) object
   (b) class
   (c) field template
   (d) method dictionary
**Vocabulary**

object, class, subclass, selector, message, method, instance variable

**Example Code**

Mitchell, p. 312-313

class name: Point
super class: Object
class var: pi
instance var: x y

class methods:
newX:xvalue Y:yvalue ||
  ^ self new x: xvalue y: yvalue
newOrigin ||
  ^ self new x: 0 y: 0
initialize ||
  pi <- 3.14159

instance messages and methods:

x: xcoord y: ycoord ||
  x <- xcoord
  y <- ycoord

moveDx: dx Dy: dy ||
  x <- dx + x
  y <- dy + y

x || ^x
y || ^y
draw || ...