Today

Why Smalltalk?

Some basics of smalltalk programs

• Syntax
• Messages
• Blocks
• Classes and Methods
• Dynamic Dispatch
• self and super

Section: The Squeak environment (projects, saving your work, etc.)
Smalltalk

- Pure object-oriented
- Class-based
- Dynamically typed

A good starting point for discussing what each of these means and what other languages look like.

The language has been quite stable since 1980.

Other points:

- A tiny language; easy to learn almost all of it
- A complete commitment to dynamic changes; little abstraction support
Overview of Smalltalk

1. All values are *objects*
   - Even numbers, code, and classes
2. Objects communicate via *messages* (handled by methods)
3. Objects have their own state
4. Every object is an instance of a class
5. A class provides behavior for its instances

This sounds a lot like Java, but smaller.

It’s also much more like Scheme than it seems; we’ll return to “what really makes something OO”

But first we need to get “the feel for Smalltalk”
Syntax

\[ \text{exp ::= atom | assign} \]

\[ \quad | \text{unarySend | infixSend | keywordSend} \]

\[ \quad | ( \text{exp } ) | \text{exp . exp | \neg exp} \]

\[ \text{atom ::= ID | literal | block} \]

\[ \text{literal ::= INTEGER | STRING | ...} \]

\[ \text{block ::= [:ID1 ... :IDn | exp] | [ exp ]} \]

\[ \text{assign ::= name := exp | name _ exp} \]

\[ \text{unarySend ::= exp ID} \]

\[ \text{infixSend ::= exp OPERATOR exp} \]

\[ \text{keywordSend ::= exp ID1: exp ... IDn: exp} \]
Some key ideas

• Really, everything is an object
• Blocks are lambdas
• Return (↑) is special
• Everything is “dynamic” – evaluation can add/remove classes, add/remove methods, etc.
• Dynamic typing
• Dynamic dispatch
• Sends to self (a special identifier; Java’s this)
Protection?

- Fields are inaccessible outside of instance
- All classes and methods are available to everyone
- No namespace management; category has no semantic significance