CSE 341: Programming Languages

Dan Grossman

Winter 2008

Lecture 19— Introduction to Ruby

Today

Why Ruby?

Some basics of Ruby programs

- Syntax
- Classes and Methods
- Variables, fields, scope
- The rep-loop, the main class, etc.

Ruby

- *Pure* object-oriented: *all* values are objects
- Class-based
- Dynamically typed
- Convenient *reflection*

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

	dynamically typed	statically typed
functional	Scheme	SML
object-oriented	Ruby	Java

Ruby vs. Smalltalk

Smalltalk, unchanged since 1980, is also pure OO, class-based, dynamically-typed.

- Smalltalk: tiny language (smaller than Scheme), elegant, regular, can learn whole thing
- Smalltalk: integrated into cool, malleable GUI environment
- Ruby: large language with a "why not?" attitude
- Ruby: scripting language (light syntax, some "odd" scope rules)
- Ruby: very popular, massive library support especially for strings, regular expressions, "Ruby on Rails"
  - Won't be our focus at all
- Ruby: *mixins* (a cool, advanced OO modularity feature)
- Ruby: blocks, libraries encourage lots of FP idioms

Really key ideas

- Really, everything is an object (with constructor, fields, methods)
- Every object has a class, which determines how the object responds to messages.
- Dynamic typing (everything is an object)
- Dynamic dispatch (focus of next lecture)
- Sends to `self` (a special identifier; Java's this)
- Everything is "dynamic" – evaluation can add/remove classes, add/remove methods, add/remove fields, etc.
- Blocks are *almost* first-class anonymous functions (later)
  - Can convert to/from real lambdas (class `Proc`)

(Also has some more Java/C like features – loops, return, etc.)

Lack of variable declarations

If you assign to a variable in scope, it's mutation.

If the variable is not in scope, it gets created (!)

- Scope is the method you are in

Same with fields: an object has a field if you assign to it

- So different objects of the same class can have different fields (!)

This "cuts down on typing" but catches fewer bugs (misspellings)

- A hallmark of "scripting languages" (an informal term)

Protection?

- Fields are inaccessible outside of instance
  - Sugar for accessor/mutator methods
  - Good OO design: subclasses can override accessors/mutators
- All classes are available to everyone
- Methods are public, protected, or private
  - protected: only callable from class or subclass object
  - private: only callable from `self`
- Namespace management, but no hiding

Unusual syntax

Just a few random things (keep your own mental list):

- Variables and fields are written differently (`@` for fields)
  - `@@` for class fields (Java's static fields)
- Newlines often matter — need extra semicolons to put things on one line
- Message sends do not need parentheses (especially with 0 arguments)
- Operators like + are just message sends
- Class names must be capitalized
- `self` is Java's `this`
- ...