**ECOOP Summer School: Teaching with Grace**

Kim Bruce
Pomona College

*Joint work with Andrew Black, James Noble, Tim Jones, Michael Homer, & a host of students.*

---

**Outline**

- Motivation for a new teaching language
- The Grace language
- Teaching with Grace
- Current status
- Homework!

---

**Why is learning to program so hard?**

- Must understand problem to be solved
- Must get logic of solution exactly correct.
- New, artificial language:
  - Must get syntax exactly right
  - Must understand semantics of language
- Ignoring issues of efficiency, reusability, etc.

---

**Can we make it easier?**

- Only to a certain extent
  - Use a restricted domain for programs
  - Provide well-constructed libraries with simple semantics
  - Provide helpful tools: syntax coloring, code-completion, IDE, debugger, ...
Language

- When tools are too complicated, focus on tools rather than product.
- Do we teach complicated power tools before screwdriver and hammer?
- Do we teach pilots to fly with an Airbus?

What if we use a language designed for novices?

... but not a toy language!

Historical Precedents

- Basic
- Logo (turtle graphics)
- Pascal
- Lesser known: Turing, Blue, ...
- Mini-worlds: Karel the Robot, Alice, Scratch, Greenfoot, ...

OO Teaching Frustrations

- Want to teach objects first
  - but must teach classes first.
- Classes are *extensible* object factories
  - Important — but not on the first day!
- Why not just define objects directly, so students can get right intuition?
Java Problems

- **public static void** `main(String[] args)`
- Primitive types *versus* object types,
  - `==` *versus* "equals"
- Flawed implementation of generics
- Static *versus* instance – on variables & methods
- float *versus* double *versus* int *versus* long

```java
>>> class aClass:
   '""A simple example class""
   val = 47
   def f(self):
       return 'hello world'

>>> x = aClass()
>>> x.value = 17
>>> x.val
47
>>> x.f()
'hello world'
```

Python Problems

```python
>>> class aClass:
   '""A simple example class""
   val = 47
   def f(self):
       return 'hello world'

>>> x = aClass()
>>> x.value = 17
>>> x.val
47
>>> x.f()
'hello world'
```

Programming is Complex

- Want students to focus on essential complexities of programming ...
- ... not accidental complexities of the language.

Is there useful user data on Programming Languages?

Controversial!
**Benefits of (Explicit) Static Types**

- Empirical studies by Stefan Hanenberg et al.
  - Static type systems help humans use a new set of classes (API)
  - Static type systems make it easier for humans to fix type errors (but not semantic errors)
  - IDE’s and documentation don’t compensate for difference w/dynamically-typed languages

_Hanenberg, invited talk, PLATEAU 2014_

**What’s important for industrial adoption of language**

- Yes:
  - Open source libs, extending existing code, familiarity
- No:
  - Simplicity, development speed

_Fruitless waiting for industry to develop simple language_

_Meyerovitch, Rabin, OOPSLA ’13_

**What if we could have:**

- Good features and low syntactic overhead of Python, _but with_
  - information hiding
  - consistent method declaration & use
  - required variable declarations
  - optional static type-checking
  - direct definition of objects

**Goal for Grace**

Integrate current ideas in programming languages into a simple, general-purpose object-oriented language aimed at helping novices learn to program.

_2010_
Target Audience

- First-year students in OO CS 1 or 2
  - objects early or late
  - static or dynamic typing
  - functionals first or scripting first or ...
- Can also be used with advanced students in OO programming course.

Introducing Grace

We are in dog-food business!

Pitch Today Aimed at Faculty

- Simple, powerful language
  - objects and classes
  - blocks provide power
  - uniform & simple syntax and semantics
- Supports variety of approaches
  - objects-early, objects-late, functional-first, ...

Hello World in Grace:

print "hello world"

Objects

def mySquare = object {
    var side := 10
    method area {
        side * side
    }
    method stretchBy(n) {
        side := side + n
    }
}

Defaults: instance variables and constants are private, methods are public - defaults can be overridden

Types

• ... are optional and can be added gradually
• ... are structural (need not be declared with object or class)
  • if it quacks like a duck, it is a duck
  • subtyping too
• Classes are not types, they are object factories!

Typed Objects

type Square = {
    area -> Number
    stretchBy (n: Number) -> Done
}

def mySquare: Square = object {
    var side: Number := 10
    method area -> Number {
        side * side
    }
    method stretchBy (n: Number) -> Done {
        side := side + n
    }
}
Classes

• ... generate objects:

```java
class aSquare.withSide (s: Number) -> Square {
    var side: Number := s
    method area -> Number {
        side * side
    }
    method stretchBy (n: Number) -> Done {
        side := side + n
    }
    print "Created square with side {s}"
}
```

No separate constructors.
Type annotations can be omitted or included

Classes

• ... abbreviate by an object with a factory method:

```java
def aSquare = object {
    method withSide (s: Number) -> Square {
        object {
            var side: Number := s
            method area -> Number {
                side * side
            }
            method stretchBy (n: Number) -> Done {
                side := side + n
            }
            print "Created square with side {s}"
        }
    }
```

Inheritance

• Single inheritance from classes or objects (perhaps with traits).
• Semantics similar to Java.
• Subtyping independent of inheritance!

Extending Types

type Graphic2D = Graphic & type {
    width -> Number
    height -> Number
    setSize (width: Number, height: Number) -> Done
    width := (width: Number) -> Done
    height := (height: Number) -> Done
}
Readability

- Multi-part method names
  - Taken from Smalltalk:
    line.from (startPoint) to (endPoint) on (canvas)

- Indenting is significant

Blocks

- Syntax for anonymous functions
  def square = {n -> n * n}
  square.apply(7) // returns 49

- Can have any number of parameters
- Represents object with apply method

Blocks

- Blocks signal delayed or repeated evaluation
  while {boolExp} do {someStuff}
  squares.forEach {n ->
    if (n.isEven) then {print n}
  }

- Blocks make it simple to define new “control structures” as methods
  method repeat (n: Number) times (block) {
    for (1 .. n) do {_: Number ->
      block.apply
    }
  }

  repeat (5) times {
    print "hi"
  }
Matching

- Provides type-safe switch/case
  ```
  match(myVal)
  case{ n: Number ->
      "The number {n+1} is next"
  }
  case{ s: String ->
      "The string {s} seen"
  }
  case{ (true) ->
      "This is true!"
  }
  ```

Avoid Hoare’s
“Billion Dollar Mistake”

- No built-in `null`
- Accessing uninitialized variable is error
- Replace `null` by:
  - sentinel objects, or
  - error actions

Sentinel Objects

A real object, tailored for the situation, e.g.:

```java
def emptyList = object {
  method asString {"<emptyList>"}
  method do(action) {} 
  method map(function) {self} 
  method size {0}
}
```

Sentinel Objects

Simplifies code, eliminates testing for `null`

```java
class aList.cons(value, tailList) {
  method asString {"({head}:{tail})"}
  method head {value}
  method tail {tailList}
  method do (action) {
    action.apply (head)
    tail.do (action)
  }
  method map (function) {
    aList.cons (function.apply (head),
    tail.map (function))
  }
  method size {1 + tail.size}
}
```
Variant Types

def absent = Singleton.named (“absent”)

type OptionNumber = Number | absent
var x: OptionNumber := table.lookUp(key)
match(x)
  case {x':Number -> ... x' ...}
  case {{(absent) -> return unknown(key)}}

*Static guarantee that x will always be matched*

val: A | B iff val:A or val:B

*Allows elimination of null*

Error Actions

*Grace encourages the use of blocks to specify error actions or default values:*

var x := table.at (key) ifAbsent{
  return unknown (key)
}

*... but also supports handling exceptions*

Dialects

*Idea “stolen” from Racket*

*Used to expand or restrict language*
  *Includes static checker.*
  *Examples:*
    * objectdraw, requiredTypes, staticTypes, *
  *Add new constructs (not new syntax)*
    *E.g., graphics primitive, control constructs, *

Dialects

*Contain a checker that can enforce constraints:*
  *All types provided, static type safety, required loop invariants/variants, pre and post-conditions, ...*

*Dialects are written in Grace*
  *... though requires knowledge of methods to extract subexpressions.*
  *Wrote a dialect to write dialects!*
**Modules**

- Are just objects
  
  ```
  import "Frog" as frogFactory
  ```

- `frogFactory` is now an object with all features defined in file `Frog.grace`

- Everything is an object!!
  - Dialects, too!

**Collections**

- Standard collections built in:
  - sequences, lists, sets, dictionaries

- Primitive arrays de-emphasized in favor of lists (like Python).

**Objectdraw Library**

- Support for
  - High-level graphics
  - Simplified event-driven programming with mouse events
  - Animations
  - GUI components

**Teaching with Grace**
Teaching with Grace

- Class tested in Fall 2014 w/ novices at Pomona College
- Class tested in Spring 2015 with seniors / graduate students at PSU.
- Graduate intake program at PSU later this summer.
- Pomona again in the fall.

Pomona Approach

- Use graphics because they are concrete
  - Add animations using timers
- Started without static types
  - Added types at end of 2nd week
  - Will move even earlier next fall
- Taught Java last 3 weeks, *alas*

Programming with Grace

Day 1: Objects

dialect "objectdraw"

object {
  inherits graphicApplication.size (400,400)

  // Make a box and display "hello world" when program begins
  filledRect.at (100 @ 200) size (50,30) on (canvas)
  text.at (90 @ 150) with ("Hello World!") on (canvas)

  // Display nested ovals and a line when mouse is pressed
  method onMousePress (point) {
    framedOval.at (140 @ 180) size (50, 40) on (canvas)
    framedOval.at (150 @ 190) size (30, 20) on (canvas)
    line.from (0 @ 400) to (400 @ 0) on (canvas)
  }

  startGraphics
}

Day 2: Using Parameters

dialect "objectdraw"

object {
  inherits graphicApplication.size (400,400)

  var nextLineStarts: Point // where mouse pressed

  // when mouse pressed remember where mouse was
  method onMousePress (point: Point) -> Done {
    nextLineStarts := point
  }

  // Draw a new line to mouse location.
  method onMouseDrag (point: Point) -> Done {
    line.from (nextLineStarts) to (point) on (canvas)
  }

  startGraphics
}

ColorScribble
First 2 Weeks

- Graphics and event-handling
  - respond to mouse events
- Conditionals
- Types
- Defining classes & objects

Weeks 3 & 4

- Declarations & Visibility
  - defs: is public
  - vars: is readable, writeable
  - methods: is confidential
- While loops and animation
  - Pong game

Weeks 5 & 6

- GUI components
  - pop-up menus, buttons, labels, text fields
  - Containers to organize objects
- Recursion
  - Recursive data structures (list & tree-like)

Weeks 7 & 8

- Lists & Matrices
  - Lists like Java ArrayList or C++ Vector
  - Access via
    - myList.at(?) or myList[7]
  - Update via
    - myList.at(?) put(“first”) or myList[7] := “first”

NestedRects

DrawingList
Weeks 9 & 10

- Inheritance
  - Single — but likely adding traits
- String algorithms
- Exceptions

Weeks 11 to 14

- Blitz intro to weirdness of Java
- I/O
- Searching & Sorting

Java Weirdnesses

- Constructors & parameters (scope)
- Location of semi-colons
- Add () for parameterless methods
- Classes/interfaces in separate files
- Private/protected/public (& default)
- Reverse order of writing types

- Multiple numeric types
- Primitive vs object types
- Required static typing
- Assignment with =
- Default values of instance variables
  - but not local variables
- null pointer exceptions

Java Weirdnesses

- Constant is “static final” or “final”
- self => this
- resolving identifiers in nested scopes: this.x
- Static overloading of methods
  - (not allowed in Grace)
- Primitive arrays
  - exceeding array bounds
- Start counting at 0
Teaching Materials

- Text: Teaching with Grace at www.cs.pomona.edu/~kim/
- Web page with previous version of course:
  - http://www.cs.pomona.edu/~kim/CSC051GF14/

Current Status

- Class tests:
  - Fall ’14 in Pomona intro. (repeat Fall ’15)
  - Spring ‘15 in o-o design course at PSU
- Implementations
  - Minigrace compiler: on web via Javascript
    - http://web.cecs.pdx.edu/~grace/minigrace/exp/
    - Also C backend, command line compiler
  - Hopper: continuation-passing interpreter
  - Kernan: interpreter in C# on Mono

Student response

- Very positive
  - Language syntax and semantics easy
  - Web-based implementation popular
- Negatives
  - Issues w/ error messages & speed,
    - though most cleared up by end of semester
  - Most negative — learning Java at end.
    - Had to transition to Java-based data structures course.

Summary

- Grace is a small yet powerful language with simple conceptual foundations
- Starting with objects simplifies teaching
  - Classes can be introduced soon thereafter
- Separating classes from types is conceptually important
- Dialects & blocks allow customization of language
- Gradual typing provides flexibility for instructors
  - add types once students have seen the need
• Please Contribute!
  • Need IDE implementors, library designers, and more.
  • Information at gracelang.org
  • Implementation at http://web.cecs.pdx.edu/~grace/minigrace/exp/
    • Use Chrome browser for best experience

Questions?
http://www.cs.pomona.edu/~kim/GraceStuff/