Teaching with Grace: First evaluations

Kim Bruce
Pomona College

Joint work with Andrew Black, James Noble, & a host of students.

Grace

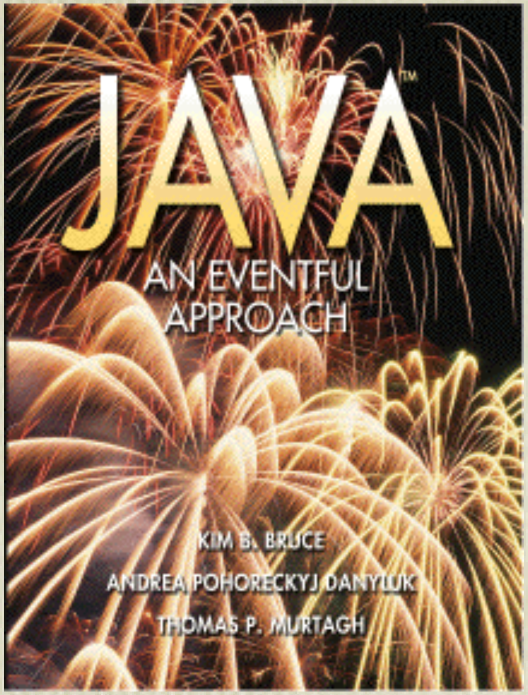
- Goal:
  - Integrate current ideas in programming languages into a simple, general-purpose object-oriented language aimed at helping novices learn to program.
- Spent 5+ years developing/implementing language (*details of language later)*

Current Status

- Implementations
  - On web via Javascript
    - http://web.cecs.pdx.edu/~grace/minigrace/exp/
  - Alternate implementation in C#
- Teaching experience so far:
  - Fall '14, '15 in Pomona intro course
  - Spring '15 in o-o design course at PSU & conversion course

Previous Intro to CS *at Pomona*

- Java-based
- Objects-first
- Event-driven programming
  - GUI
- Graphics
- Animations using threads
- Text is Java: An eventful approach



The Experiment

- Rewrite text for Grace:
  - Programming with Grace
- Teach new Grace section in parallel with existing Java sections.
- Presented as experimental section that would teach Java by end.

Course Structure

- 10 weeks (29 lectures) of Grace
  - Objects, classes, control structures, recursion, inheritance & subtyping, strings, exceptions, *graphics*, *animations (concurrency)*, *GUI event-handling, lists*, *matrices.*
- 4 weeks of Java
  - including threads, arrays, I/O
  - Searching, sorting

Courses Matched

- Texts *(rewrite of Java approach to Grace)*
- Programming assignments
  - Including test programs
- Exams
- Major difference: learning 2nd language

Student response to Grace

- Very positive
    - Language syntax and semantics easy.
    - Web-based implementation popular
- Negatives
    - Issues w/ error messages & compile speed,
        - execution speed fine
    - Most negative — learning Java at end.
        - Had to transition to Java-based data structures course.

Preliminary Results

- Grace class did better or equal to Java in every measure:
  - Midterm: median +9, mean +14
  - Final: median 0, mean +1 *Why?*
  - Test Program 1: median +4, mean +4
  - Test Program 2: median +8, mean +4
    - *due 2 weeks earlier for Grace students*

What's wrong with current languages?

Why go to this effort?

Java Problems

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

Python Problems

```
>>> 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'
```

disappearing self?

uncaught typos

no information hiding
except by name mangling

Grace overview in 2 slides

- Object-based (*with classes*)
- First-class closures (*look like blocks*)
  - Everything is an object
- Default visibility is "correct"
- Multi-part method names
- Indenting is significant (*but braces too*)

Grace overview in 2 slides

- Single numeric type
- Gradually typed (*gradually*)
    - Structural types distinct from classes
- No null (*use match/variant types*)
- Lists rather than arrays
- Dialects

Hello World in Grace:

```
print "hello world"
```

Objects

```
def mySquare = object {
      def smallest = 2

      var side := 10

      method area {
           side * side
      }

      method stretchBy(n) {
           side := side + n
      }
}
```

Defaults: defs, variables & constants are confidential,
methods are public - 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!*

Classes in Grace

- ... generate objects:

```
class aSquareWithSide (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}"
}
```

Create object with aSquareWithSide(20)

No separate constructors.

Type annotations can be omitted or included

Classes in Java

```
public class SquareWithSide implements Square {
    private int side;

    public SquareWithSide(int s) {
        side = s;
        System.out.println( "Created square with side” + s);
     }

    public int area() {
       return side * side;
    }

    public void stretchBy (int n) {
        side = side + n;
    }
}
```

Create object with
new SquareWithSide(20)

Side by Side

```
class aSquareWithSide (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}"
}
```

```
public class SquareWithSide implements Square {
    private int side;

    public SquareWithSide(int s) {
         side = s;
         System.out.println( "Created square with s
      }

    public int area() {
       return side * side;
    }

    public void stretchBy (int n) {
         side = side + n;
    }
 }
```

Multi-part method names

- Taken from Smalltalk
- Makes code more readable:

```
lineFrom (startPoint)
      to (endPoint) on (canvas)
```

- *Indenting is significant*

Blocks

- Syntax for anonymous functions

```
def square = {n -> n * n}   ←   function
square.apply (7)  // returns 49

def nums = 1 .. 100
def squares = nums.map {n -> n * n}
```

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

Blocks

- Blocks make it simple define new “control structures” as methods

```
method repeat (n: Number) times (block) {
  for (1 .. n) do {i: Number ->
    block.apply
  }
}

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

while {b} pausing (ms) do {code}

Avoid Hoare's "Billion Dollar Mistake"

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

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, ...

Advantages over Java

- Use objects as programs, classes later
  - no public static void main
- Only 1 numeric type
- No separate constructor “method”
- Blocks as listeners for GUI
- Use lists instead of arrays
- No “equals” method, no overloading

Advantages over Java

- No classes as types, no “static” features
  - no primitive types
- Simple (*modern*) for loops
- Use loops with timers instead of Threads
- No null pointer exception
  - *uninitialized error instead*
- Type-safe match instead of casts

Java has, but Grace does not

1 Type-based overloading of methods.
2 Arity-based overloading.
3 Primitive data — int, boolean, char, byte, short, long, float, double.
4 Classes (as built-in non-objects).
5 Packages (as built-in non-objects).
6 Constructors (as distinct from methods) and new.
7 Object initializers (code in a class enclosed in { and })
8 import * — introduction of names invisibly.
9 Operations on variables, like x++ meaning x := x + 1.
10 Multiple numeric types (so that, for example, 3.0 and 3 are different).
11 Numeric literals with F and L.
12 Integer arithmetic defined to wrap.
13 == as a built-in operation on objects.
14 static variables.
15 static methods.

Java has

16 static initializers.

17 final.

18 private (which is much more complicated than most people realize, since it interacts with the type system).

19 C-style for loops.

20 switch statements.

21 Class-types.

22 Packages

23 Package-based visibility.

24 Arrays (as a special built-in construct with their own special syntax and type rules).

25 Required semicolons.

26 () in method requests that take no parameters.

27 public static void main(String[] args)   necessary to run your code.

28 Object with "functional interfaces" treated as λ-expressions.

29 Null

Grace has

1. String interpolation “The value of x is {x}”
2. Object expressions
3. Nested objects
4. Closures w/correct scope
5. Operators defined as methods
6. Match statements & variant types

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

Grace

- Please Contribute!
  - Need IDE implementors, library designers, and more.
  - Want to teach with it?
  - Information at gracelang.org
  - Implementation at http://web.cecs.pdx.edu/~grace/minigrace/exp/
    - *Use Chrome browser for best experience*

Questions?