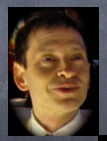
Designing *Grace*

Can an Introductory Programming Language Support the Teaching of Software Engineering?


James Noble


Andrew Black


Michael Homer


Kim Bruce

gracelang.org

Why Now?

- Happy teaching Java next 3-5 years
- In 2015, Java will be 20 years old
  - Java 8 is coming...
- State of the art has advanced
  - patches look like ... patches
- Essential difficulties vs Accidental difficulties
- To be ready in 2015, we need to start now.

Grace User Model

- First year students in OO CS1 or CS2
  - objects early or late,
  - static or dynamic types,
  - functionals first or scriptings first or ...
- Second year students
- Faculty & TAs — assignments and libraries
- Researchers wanting an experimental vehicle
- Language Designers wanting a good example

Grace Example

```
method average(in : InputStream) -> Number
// reads numbers from in stream and averages them
{  var total := 0
   var count := 0
   while { ! in.atEnd } do {
      count := count + 1
      total := total + in.readNumber }
   if (count == 0) then {return 0}
   return total / count }
```

Method Requests

```
aPerson.printOn(outputStream)

printOn(outputStream)   // implicit self

((x + y) > z) && !q        // operators are requests

while { ! in.atEnd } do { print (in.readNumber) }
                          // multi-part method name
```


Object constructors

```
object {
   def x = 2
   def y = 3
   method distanceTo(other) {
      ((x - other.x)^2 + (y - other.y)^2) }
}
```


Object constructors

```
object {
   def x = 2
   def y = 3
   method distanceTo(other) {
      ((x - other.x)^2 + (y - other.y)^2) }
}
```

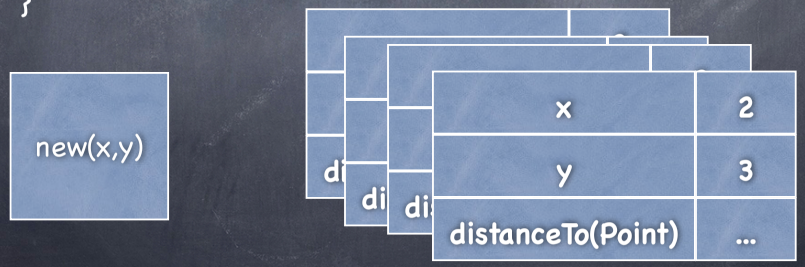
Object constructors

```
object {
   def x = 2
   def y = 3
   method distanceTo(other) {
      ((x - other.x)^2 + (y - other.y)^2) }
}
```

x	2
y	3
distanceTo(Point)	...

Classes

```
class CartesianPoint.new(x', y') {
    def x = x'
    def y = y'
    method distanceTo(other) {
        ((x - other.x)^2 + (y - other.y)^2) }
}
```



Can an Introductory Programming Language Support the Teaching of Software Engineering?

Consistency

- Syntactic Consistency:

```
if (count == 0) { return 0 }          // C/Java
if (count == 0) then { return 0 }     // Grace
```


Consistency

- Syntactic Consistency:

```
if (count == 0) { return 0 }          // C/Java
if (count == 0) then { return 0 }     // Grace
```

- Semantic Consistency:

```
while ( x > 0 ) { other.iterate }     // C/Java
while { x > 0 } do { other.iterate }  // Grace
```

Static vs. Dynamic Types

```
class CartesianPoint.new(x', y' )
{
  def x = x`
  def y = y'
  method distanceTo(other) {
    ((x - other.x)^2 + (y - other.y)^2) }
}
```


Static vs. Dynamic Types

```
class CartesianPoint.new(x' : Number, y' : Number) -> Point
{
  def x : Number = x`
  def y : Number = y'
  method distanceTo(other : Point) -> Number {
    ((x - other.x)^2 + (y - other.y)^2) }
}
```


Types vs Classes

```
type Point = {
  x -> Number
  y -> Number
  distanceTo(other : Point) -> Number
}
```

- Types are separate from classes
- Types need to be defined separately

Implicit vs. Explicit Declarations

- JavaScript, FORTRAN:

```
countr = counter + delta
```

- Pascal, C, Java, Ada...

```
def delta = 3
var counter := 0

counter := counter + delta
if (counter == 100) then { ... }
```

Information Hiding

```
def joe = object {
    var forename := "Joe"
    var surname := "Bloggs"
    var id := 234567
    method asString
            {"Name: {forename} {surname} Id: {id}"}
}

print "joe is {joe}."  // error here
```


Information Hiding

```
def joe = object {
    var forename is public, readable := "Joe"
    var surname is public, readable := "Bloggs"
    var id := 234567
    method asString is public
            {"Name: {forename} {surname} Id: {id}"}
}

print "joe is {joe}."  // works now
```


Formal Reasoning

```
method gcd(m, n) {
    assert {(m >= 0) & (n >= 0) & ((m != 0) | (n != 0))}

    var a := max(m,n)
    var b := min(m,n)
    while {b != 0}
      invariant { a >= b }
      do {def remainder = a % b
          a := b
          b := remainder
      variant {b}
    return a }
```


Formal Reasoning

```
assert {(letters.size > 0) && (letters.size < 20)}

// implementation of assert
method assert (block : Block<Boolean>) {
  if ( ! block.apply )
    then {error "Assertion Failed"}
}
```

Formal Reasoning

```
method for(collection) invariant(inv) do(blk) {

  for (collection) do {element->
    if (! inv.apply) then {
      InvariantFailure.raise "Loop invariant not satisfied." }
    blk.apply(element)
    }

  if (! inv.apply) then {
    InvariantFailure.raise "Loop invariant not satisfied." }

}
```


Dialects

```
dialect "loopinvariant"
import "mgcollections" as collections

def data = collections.list.new(2, 3, 4, 5)

var sum : Number := 0

for (data) invariant { sum >= 0 } do
            { item : Number -> sum := sum + item }
```


Can an Introductory Programming Language Support the Teaching of Software Engineering?

- *Grace*: objects and method requests
- Consistency: syntactic vs semantics
- Static vs Dynamic Types
- Types vs Classes
- Information Hiding
- Formal Reasoning
- Dialects

No conclusions — we aren't done yet

Questions

Comments

Suggestions

Brickbats

Help!

- Supporters
- Programmers
- Implementers
- Library Writers
- IDE Developers!!!!
- Testers
- Teachers
- Students
- Tech Writers
- Textbook Authors
- Blog editors
- Community Builders

http://gracelang.org