Grace

An open-source educational OO language

Michael Homer

Victoria University of Wellington

Andrew Black

Kim Bruce

James Noble

Grace user model

- First-year students in OO CS1 or CS2
  - objects early or late
  - static or dynamic types
  - functions first or scripting first or . . .
- Second-year students
- Faculty and TAs – assignments and libraries

Principles

- Simple programs should be simple
- Understandable semantic model
- Be a general-purpose language
- Optional types
- Support different teaching orders

Incantations

```
package user;

class HelloWorld {
    public static  void main(String args[])  {
        System.out.println("Hello world");
    }
}
```

No incantations

```
print "Hello world"
```

Scripting first

```
var fib := 1
var fibprev := 1
for (3..7) do {i->
    def tmp = fibprev
    fibprev := fib
    fib := fib + tmp
}

print "7th Fibonacci number: {fib}"
```

Objects first

```
def FibonacciCalculator = object {
    var current := 1
    var previous := 1
    var upTo := 1
    method next {
      upTo := upTo + 1
      current := current + previous
      previous := current — previous
      return current
    }
    method printNext {
      print "{upTo+1}th Fibonacci number: {next}"
    }
}
```

With types

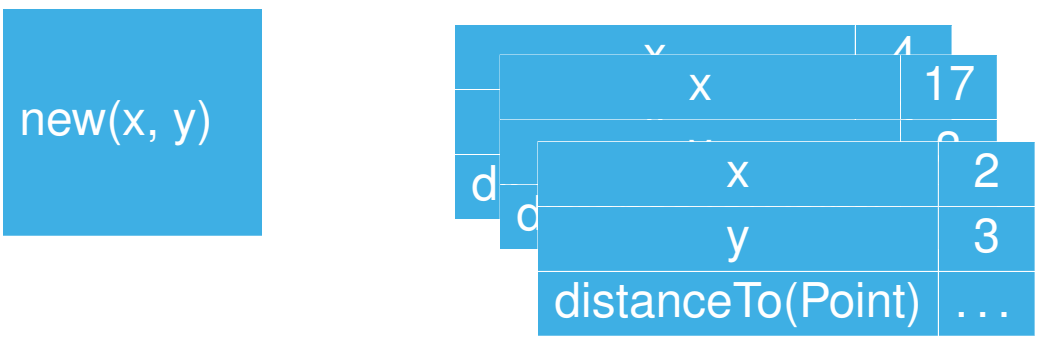
```
def FibonacciCalculator = object {
    var current : Number := 1
    var previous : Number := 1
    var upTo : Number := 2
    method next -> Number {
      upTo := upTo + 1
      current := current + previous
      previous := current - previous
      return current
    }
    method printNext -> Done {
      print "{upTo+1}th Fibonacci number: {next}"
    }
}
```

Classes first

```
class FibonacciCalculator.new(i1, i2) {
    var current := i1
    var previous := i2
    var upTo := 2
    method next {
        upTo := upTo + 1
        current := current + previous
        previous := current - previous
        return current
    }
    method printNext {
        print "{upTo+1}th Fibonacci number: {next}"
    }
}
```

Classes are factories

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



Classes are objects

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

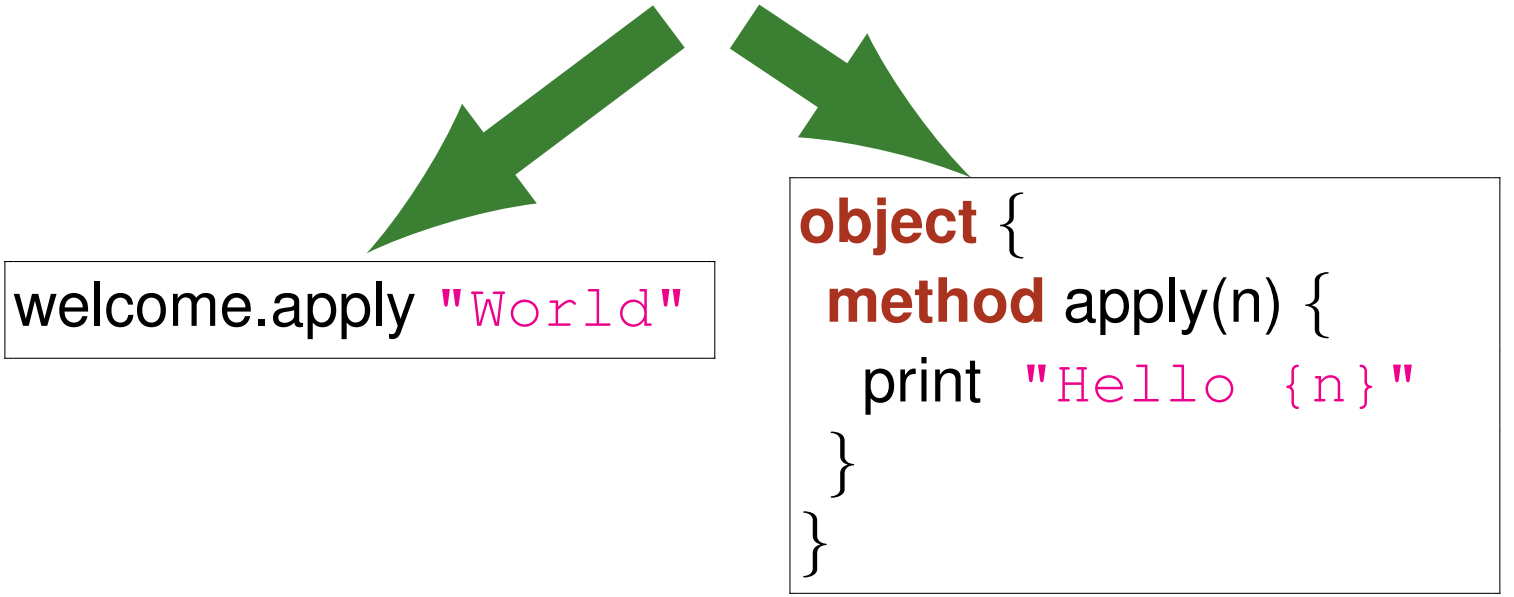
Method requests

```
people.add(person)

print "Hello, world!"
  // Implicit receiver

((x + y) > z) && !q
  // Operators are methods

5.between(3)and(8)
  // Multi-part method name

obj.x := 2
  // Accessor methods
```

Blocks

Are objects:

```
def welcome = { n -> print "Hello {n}" }
```



Control structures

```
if (count == 0) then {
    print "NONE"
} else {
    print "SOME"
}

while {i > 0} do {
    print "{i} bottles of beer on the
      wall"
    i := i - 1
}
```

Control structures as methods

```
method if(c : Boolean) then (t : Block) else (
    f : Block) {
    c.ifTrue(t)else(f)
}

method while(c : Block) do(a : Block) {
    c.apply.ifTrue {
        a.apply
        while (c) do (a)
    }
}
```

Visibility

```
def pt = object {
  var x := 2
  var y is readable := 3
  var z is public, readable, writable := 4
}

pt.x // No such method

pt.y // Requested confidential method

pt.z := pt.z + 1 // OK
```

Types

- Types are for grouping
  - Optional
  - Gradual
  - Structural

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

Generics

```
type List<X> = {
    add(item : X) -> Done
    get(index : Number) -> X
}
```

Generics

```
type List<X> = {
    add(item : X) -> Done
    get(index : Number) -> X
}
type ReadOnlyList<X> = {
    get(index : Number) -> X
}
```

No null pointer exceptions!

Type operations

- Variants: Point | Nil

  $x : A \mid B \equiv x : A \vee x : B$

```
def nilValue : Nil = ...
var p : Point | Nil := nilValue // OK
 ...
p := CartesianPoint.new(3,4) // OK
```

- Intersection: T1 & T2 conforms to T1 and T2

Pattern matching

```
match(x) // x : 0 | String | Student

  // Match against a literal
  case { 0 -> print "Zero" }

  // Typematch, binding a variable
  case { s : String -> print(s) }

  // Destructuring match
  case { _ : Student(name, id)-> print(name)}
```

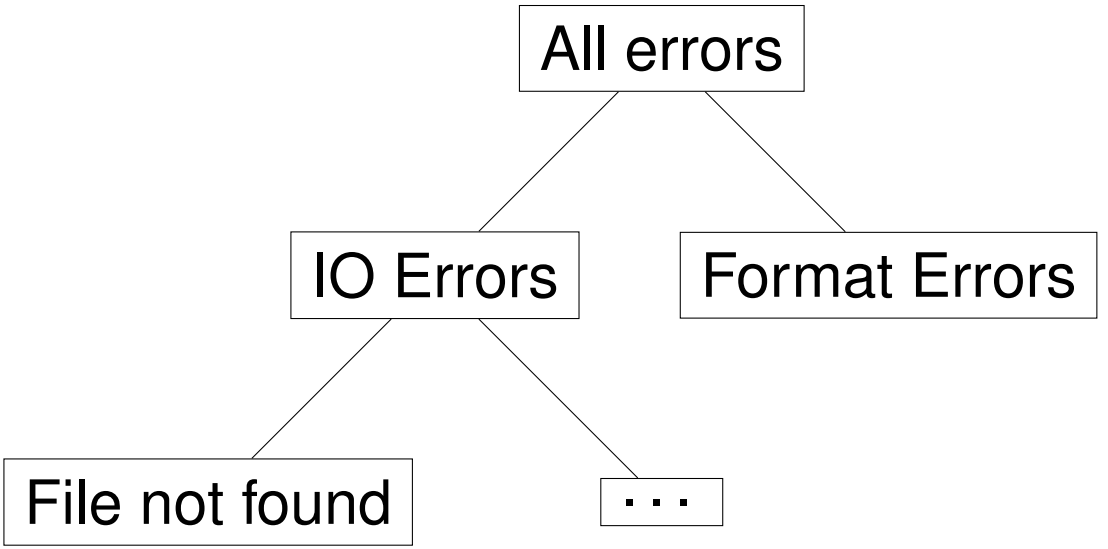
Destructurable objects

```
class CartesianPoint.new(x', y') {
    def x = x'
    def y = y'
    method extract {
        [x, y]
    }
}

type Point = {
    x -> Number
    y -> Number
    extract -> Tuple<Number, Number>
}
```

Patterns via requests

```
if (Point.match(x)) then {
    ...
}

def Odd = object {
    method match(o) {
        if ((o % 2) == 1) then {
            SuccessfulMatch.new(o)
        } else {
            FailedMatch.new(o)
        }
    }
}
```

Exceptions



- Useful!
- Bubble up to find a handler

Exceptions as patterns

```
def MyError = Error.refine "MyError"
def NegativeError = MyError.refine "
  NegativeError"

catch {
    if (value < 0) then {
      NegativeError.raise "{value} < 0"
    }
} case {e : MyError -> print "Error: {e}"}
```

Implementation

- Written in Grace
- Supports almost all of the dynamic language
- Static typechecking present but limited

Compiler source code:
`https://github.com/mwh/minigrace`

Tarballs:
`http://ecs.vuw.ac.nz/~mwh/minigrace/`

Client-side web front-end:
`http://ecs.vuw.ac.nz/~mwh/minigrace/js/`

No conclusions – we aren't done yet

Questions

Comments

Suggestions

Brickbats

Help!

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

Grace

An open-source educational OO language

Michael Homer

Victoria University of Wellington

Andrew Black

Kim Bruce

James Noble

Contents