Building A Graceful Language

Design by Instructor

Timothy Jones

Victoria University of Wellington

tim@ecs.vuw.ac.nz

December 16, 2013

The Language

Frustration with languages used for teaching

Pascal is old, Java is bloated

> *Grace is the absence of everything that indicates pain or difficulty, hesitation or incongruity.*
>
> — William Hazlitt

Goals

- Support multiple paradigms
  - Objects
  - Scripting/Procedural
  - Functional
- Minimise conceptual burden
- Diverse applications within syntactically consistent language

First Taste

In Java:

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

In Grace:

```
print "Hello world"
```

Objects

```
var john := object {
  def name is public = "John"

  method say(phrase) {
    print "John says {phrase}"
  }

  print "John has been born!"
}
```

Gradually Typed

```
type Person = {
  name → String
  say(phrase : String) → Done
}

var kate : Person := object {
  def name : String is public = "Kate"

  method say(phrase : String) → Done {
    print "Kate says {phrase}"
  }
}
```

Classes

```
class aPerson.named(name′) → Person {
    def name is public = name′
    method say(phrase) { print "{name} says {phrase}" }
}
```

Translates to:

```
def aPerson = object {
    method named(name′) → Person {
        object {
            def name is public = name′
            method say(phrase) { print "{name} says {phrase}" }
        }
    }
}
```

Blocks

First class functions:

```
def double = { x → x + x }
double.apply 10
```

Like numbers and strings, no need for parens in method requests

```
method shiftUp(list : List⟨Number⟩) {
  list.map { x → x + 1 }
}
```

Control Structures

Methods may be written 'mixfix'

```
method substringFrom(start) to(end) { ... }
str.substringFrom 3 to 5
```

Combined with blocks, we can define our own control structures:

```
method while(cond) do(block) {
  if(cond.apply) then {
    block.apply
    while(cond) do(block)
  }
}

while { x < y } do { x := x * 2 }
```

Dialects

Change the local definitions (but not the syntax) of a module

Check that the module conforms to certain rules (eg. must use types, no mutable variables)

The entire static type system is just a dialect!

The Designers

- Andrew Black
- Kim Bruce
- James Noble

The Designers

- Object Constructors and Dynamic Typing
- Kim Bruce
- James Noble

The Designers

- Object Constructors and Dynamic Typing
- Classes and Static Typing
- James Noble

The Designers

- Object Constructors and Dynamic Typing
- Classes and Static Typing
- The Mediator

The Audience

Designed by Instructors, for Instructors

Differences of opinion in the design represent real-world differences of opinion in how OO should be taught

Compromise leads to interesting design decisions!

Implementing Inheritance

Objects-only? Delegation!

```
def snake = object {
    def noise = "hiss"
    method makeNoise {
        print(self.noise)
    }
}

def rattleSnake = object {
    inherits snake
    def noise = "rattle"
}
```

Implementing Inheritance

Objects-only? Delegation!

```
def snake = object {
    def noise = "hiss"
    method makeNoise {
        print(self.noise) // self is bound to the receiver of makeNoise
    }
}

def rattleSnake = object {
    inherits snake
    def noise = "rattle"
}
```

The Identity Problem

The two objects have separate identities!

```
def snake = object {
    def this = self
    def noise = "hiss"
    method makeNoise {
        print(this.noise)
    }
}

def rattleSnake = object {
    inherits snake
    def noise = "rattle"
}
```

‘Becomes’ Inheritance

Solution: an inheriting object merges identities with its super object

```
class aSnake.new {
    def this = self
    def noise = "hiss"
    method makeNoise {
        print(this.noise) // identity of self is rewritten to be rattleSnake
    }
}

def rattleSnake = object {
    inherits aSnake.new // can only inherit from a fresh object
    def noise = "rattle"
}
```

Initialisation Problem

Objects have different structure at each part of the constructor chain

```
class aSnake.new {
    def noise = "hiss"
    self.makeNoise // self is not yet rattleSnake
}

def rattleSnake = object {
    inherits aSnake.new
    def noise = "rattle"
    method makeNoise {
        print(self.noise)
    }
}
```

Constructor Specialisation

Every method with a tail-call object has two variants

```
method new {
  object {}
}

method new_inherits(self) {
  ...
}
```

Essentially JavaScript's **new** Snake vs. Snake.call(**this**)

Abstract methods

```
class aBird.new {
    method fly {
        if(self.canFly) then {
            print "take off!"
        } else {
            print "crashed!"
        }
    }
}

def kiwi = object {
    inherits aBird.new
    def canFly = false
}
```

Abstract methods

```
class aBird.new { // is this class well-typed?
    method fly {
        if(self.canFly) then {
            print "take off!"
        } else {
            print "crashed!"
        }
    }
}

def kiwi = object {
    inherits aBird.new
    def canFly = false
}
```

Typing Self

What is the type of **self**?

The value is never explicitly given a type, so how do you supply it?

Typing Self

What is the type of **self**?

The value is never explicitly given a type, so how do you supply it?

Default is dynamic, add extra information in a dialect

Typing Self

What is the type of **self**?

The value is never explicitly given a type, so how do you supply it?

Default is dynamic, add extra information in a dialect

Add annotations to prevent all-or-nothing scenario

How far away are we?

- Trial courses starting next year
- Tooling and development environments the next major goal
- Ready for general consumption by 2015?

Links

gracelang.org

ecs.vuw.ac.nz/~mwh/minigrace/js

github.com/mwh/minigrace

grace-core@cecs.pdx.edu