

Delegation isn't quite Inheritance

Objects, Classes and Initialisation in Grace

James Noble

Victoria University of Wellington

New Zealand

kjx@ecs.vuw.ac.nz

Abstract

Inheritance and delegation are often considered roughly equivalent, but their initialisation semantics are very different. Grace's hopes to support both object- and class-based programming is greatly complicated by this difference.

1. Objects

Objects in Grace (an object-oriented, block-structured, gradually- and structurally-typed language) are created ex nihilo by *object constructors* [1, 2, 4, 6, 10].

```
def amelia = object {
  inherits cat.new("Amelia")
  def question = answer
  var capacity := 3
  print "The answer is {question}"
  method answer {39 + capacity}
}
```

Object constructors can **define** constants, **variables**, and **methods**: code initialising constants and methods is executed when it is encountered in the constructor body, along with any other inline code (the “print” statement.)

2. Inheritance from Classes

Grace aims to support classes as well as objects. The object above inherits from the `cat` class's sole factory method `new`.

```
class cat.new(name : String) {
  def answer is public = name
  print "New cat {self}"
}
```

Initialisation occurs in the context of the final object: `self` is bound to `amelia`; the `answer` method overrides the `cat`'s. This supports a number of common programming idioms, but also causes a range of problems. Pragmatically, this imports many of the initialisation and undefinedness problems common to Java and its successors [5, 12, 15].

Conceptually, Grace claims to be an object-oriented language, one that can be understood without classes, but this kind of inheritance and initialisation is not easily explicable purely in terms of objects (why is `cat`'s `answer` method somehow overridden?) and the explanation is not compositional (writing “`cat.new("Amelia")`” in straight-line code creates a new `cat` object: why should an `inherits` clause change that?)

3. Delegation from Objects

Delegation can avoid the conceptual problems of classes and inheritance — objects can be understood without classes, and inherited (parts of) objects stand alone and are created in exactly the same whether or not inherited [7–9, 11, 14]. On the other hand, programming patterns (like the explicit `self` in `cat`) will not bind to the “final” inheriting object; so debuggers must make the multiple super-part-objects explicit. Precisely because delegation borrows behaviour from preexisting objects that become super-parts of the final object, the preexisting objects can never be initialised in the context of the final object: every object has its own identity. Indeed, because they are preexisting, super-part objects can become “shared parts” of many different final objects [3]. Concatenation (by value where delegation is by reference) has similar benefits and problems due to preexisting super-part objects [13].

4. Traits

These problems can also be resolved by permitting delegation (or inheritance) only from traits — objects (or classes) without mutable state, without initialisation, and that do not explicitly or implicitly capture `self`. Unfortunately these restrictions make trait declarations quite different from object constructors: more different than class declarations.

References

- [1] BLACK, A. P., BRUCE, K. B., HOMER, M., AND NOBLE, J. Grace: the absence of (inessential) difficulty. In *Onward!* (2012), pp. 85–98.
- [2] BLACK, A. P., JUL, E., HUTCHINSON, N., AND LEVY, H. M. The development of the Emerald programming language. In *History of Programming Languages III* (2007), ACM Press.
- [3] CHAMBERS, C., UNGAR, D., CHANG, B.-W., AND HÖLZLE, U. Parents are shared parts of objects: inheritance and encapsulation in Self. *Lisp and Symbolic Computation* 4, 3 (1991).
- [4] DEDECKER, J., CUTSEM, T. V., MOSTINCKX, S., D’HONDT, T., AND MEUTER, W. D. Ambient-oriented programming in AmbientTalk. In *ECOOP* (2006), pp. 230–254.
- [5] GIL, J. Y., AND SHRAGAI, T. Are we ready for a safer construction environment? In *ECOOP* (2009).
- [6] IERUSALIMSCHY, R., DE FIGUEIREDO, L. H., AND CELES, W. The evolution of Lua. In *HOPL-III* (2007).
- [7] LIEBERMAN, H. Using prototypical objects to implement shared behavior in object oriented systems. In *OOPSLA* (Nov. 1986).
- [8] LIEBERMAN, H., STEIN, L. A., AND UNGAR, D. Treaty of Orlando. In *Addendum to OOPSLA Proceedings* (May 1988).
- [9] NOBLE, J., TAIVALSAARI, A., AND MOORE, I., Eds. *Prototype-Based Programming: Concepts, Languages and Applications*. Springer-Verlag, 1999.
- [10] SEVERANCE, C. JavaScript: Designing a language in 10 days. *IEEE Computer* 45, 2 (Feb. 2012), 7–8.
- [11] STEIN, L. A. Delegation is inheritance. In *OOPSLA* (Dec. 1987).
- [12] SUMMERS, A. J., AND MÜLLER, P. Freedom before commitment: a lightweight type system for object initialisation. In *OOPSLA* (2011).
- [13] TAIVALSAARI, A. Delegation versus concatenation or cloning is inheritance too. *OOPS Messenger* 6, 3 (1995).
- [14] UNGAR, D., AND SMITH, R. B. SELF: the Power of Simplicity. *Lisp and Symbolic Computation* 4, 3 (June 1991).
- [15] ZIBIN, Y., CUNNINGHAM, D., PESHANSKY, I., AND SARASWAT, V. Object initialization in X10. In *ECOOP* (2012).