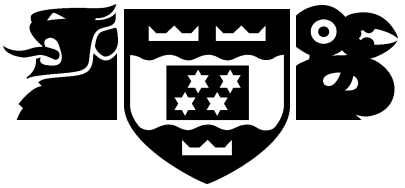
VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui



School of Mathematical and Computing Sciences
Computer Science

Oh! Gee! Java!
Ownership types (almost) for free

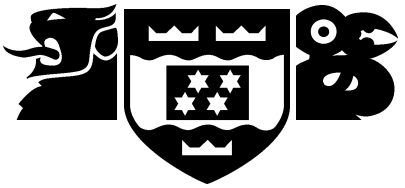
James Noble and Robert Biddle

Technical Report CS-TR-03/9
May 2003

School of Mathematical and Computing Sciences
Victoria University
PO Box 600, Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Email: Tech.Reports@mcs.vuw.ac.nz
http://www.mcs.vuw.ac.nz/research

VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui



School of Mathematical and Computing Sciences
Computer Science

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341, Fax: +64 4 463 5045
Email: Tech.Reports@mcs.vuw.ac.nz
http://www.mcs.vuw.ac.nz/research

Oh! Gee! Java!
Ownership types (almost) for free

James Noble and Robert Biddle

Technical Report CS-TR-03/9
May 2003

Abstract

The existing proposals for ownership types build on Java-like languages without generic types. A minimal ownership type scheme for languages with generic types could be a much smaller change to the language. Given that Java-like languages are now adopting generic types (and that several object-oriented languages already provide them) such an extension may make ownership types practical for the practising programmer.

Author Information

James Noble and Robert Biddle are members of the academic staff in the School of Mathematical and Computing Sciences at Victoria University of Wellington.

Oh! Gee! Java!

Ownership types (almost) for free

James Noble[1], Robert Biddle[1], and et. al.

School of Mathematical and Computing Sciences, Victoria University of Wellington

Abstract. The existing proposals for ownership types build on Java-like languages without generic types. A minimal ownership type scheme for languages with generic types could be a much smaller change to the language. Given that Java-like languages are now adopting generic types (and that several object-oriented languages already provide them) such an extension may make ownership types practical for the practising programmer.

1 Introduction

The existing proposals for ownership types [5, 1, 8, 3, 2] build on Java-like languages without generic types. These languages are changing to include generics, however, and there are a number of "Java-unlike" languages (C++, Eiffel, Ada-9X) that already include generic types.

Our position is that a minimal ownership type scheme should be a smaller extension to these languages than to languages without generics, and that such an extension may make ownership types practical for the practising programmer.

2 Questions

This position raises three questions. We do not presume to have complete *answers* to these questions at this stage, but we consider these questions worthy of discussion:

1. What is the minimal set of extensions needed to a generic type scheme to provide ownership types?
2. What ownership model should be adopted by such a scheme?
3. How should such a scheme be configured to fit in with existing language constructs?

3 Oh! Gee! Java!

Oh! Gee! Java [7, 6] is our one-page outline of an ownership scheme for generic-extended Java [4]. This is a straw-man proposal for discussion: we make no claims for completeness, consistency, or correctness.

3.1 Extensions for Ownership

The main extension we propose is that all objects have an extra type parameter representing their *owner*. This parameter is the *last* entry in the type parameter list as it is optional in both declaration and use of a class or interface name. In a declaration, the owner parameter must use the type variable name `owner`. If omitted, the owner parameter defaults to `public` (i.e. unrestricted/shared/world ownership).

Expressions whose ownership does not match are not assignment compatible. Note that as every object and variable will be `public` by default, this restriction has no effect unless other values are established for ownership parameters.

3.2 Ownership Model

We propose the following contexts for ownership to match existing program structures in Java:

private An object owned by `this` and which may only be accessed by methods of the current class.
protected: An object owned by `this`.
static An object shared between all instances of a class (like a Type Universe [8]).
package An object shared within a package (like a Confined Type [2]).
public An object shared throughout the whole program.

A parameterised type instantiation may use one of these names, or any type parameter in scope, as an actual type argument to bind the ownership parameter. We have chosen these names simply to introduce no new Java keywords. At this stage, we are agnostic as to whether inner classes should have the same access rights as their containing object [3]

3.3 Configuration

We use existing generic type parameters to carry ownership, as outlined above, and introduce no new keywords. Our ownership region has been chosen to model the existing protection structures of Java, including per-instance, per-class, and per-package ownership. As in generic Java, our goals for this extension are to be completely upwardly compatible with Java and thus to accept all existing Java (and Generic Java) programs, while providing support for ownership for those programmers who wish to use it.

4 A Generic Example

This is a simple linked-list example, adapted from [1].

```
class StackClient {
  Stack<Integer<private>, private> st = new Stack<Integer<private>, private>;
    // stack itself is owned locally, as are the Integers it contains

  void run() {
    Integer<private> p = new Integer<private>(5);
    st.push(p);
    Integer<private> p2 = st.pop():
      // no casts required due to type and ownership parameterisation.
  }

  Stack<Integer, private> hole1() { return st; }
    // error if called outsite Stackclient, breaks private ownership
}


class Stack<Element> {   // Owner never appears, so can be omitted!
  private Link<Element, private> top;

  public Element pop() {
    if (top == null)
      return null;
    Link<Element, private> temp = top;
    top = top.next();
    return temp.member();
  }
  public void push(Element o) {
    top = new Link<Element, private> (o, top);
  }
}

class Link<Element, Owner> {
  private Link<Element, Owner> nxt; // same Owner as this!
  private Element obj; // both type and ownership from Element parameter

  public Link<Element, Owner>(Element _obj, Link<Element, Owner> _nxt) {
   obj = _obj; nxt = _nxt;
  }
  public Link<Element, Owner> next() { return nxt; }
  public Element member() { return  member; }
}
```

References

1. J. Aldrich, V. Kostadinov, and C. Chambers. Alias annotations for program understanding. In *OOPSLA Proceedings*, November 2002.
2. Boris Bokowski and Jan Vitek. Confined types. In *OOPSLA Proceedings*, 1999.
3. Chandrasekhar Boyapati, Barbara Liskov, and Liuba Shrira. Ownership types for object encapsulation. In *ACM Symposium on Principles of Programming Languages (POPL)*, January 2003.
4. Gilad Bracha, Norman Cohen, Christian Kemper, WSteve Marx, Martin Odersky, Sven-Eric Panitz, David Stoutamire, Kresten Thorup, and Philip Wadler. Adding generics to the java programming language: Participant draft specification. Technical Report JSR-14, Sun Java Community Process, April 2001.
5. Dave Clarke and Sophia Drossopoulou. Ownership, encapsulation, and the disjointness of type and effect. In *OOPSLA Proceedings*, 2002.
6. Doug Cooper. *Oh! My! Modula-2!* Norton, 1991.
7. Doug Cooper and Michael Clancy. *Oh! Pascal!* Norton, 1982.
8. P. Müller and A. Poetzsch-Heffter. A type system for controlling representation exposure in Java. In *ECOOP Workshop on Formal Techniques for Java Programs*, number TR 269 in Technical Report. Fernuniversität Hagen, 2000.