VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wānanga o te Ūpoko o te Ika a Māui

School of Engineering and Computer Science
Te Kura Mātai Pūkaha, Pūrorohiko

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Internet: office@ecs.vuw.ac.nz

An Empirical Study of Delegation vs. Inheritance

Luke Inkster

Supervisors: Alex Potanin, James Noble & Tim Jones

Submitted in partial fulfilment of the requirements for
Bachelor of Engineering with Honours.

Abstract

A short description of the project goes here.

Acknowledgments

Any acknowledgments should go in here, between the title page and the table of contents. The acknowledgments do not form a proper chapter, and so don't get a number or appear in the table of contents.

Contents

Figures

Chapter 1

Introduction

This chapter gives an introduction to the project report.

In Chapter 2 we explain how to use this document, and the `vuwproject` style. In Chapter **??** we say some things about LaTeX, and in Chapter 3 we give our conclusions.

Chapter 2

Literature Review

2.1 Are we Ready for a Safer Construction Environment [3]

This paper by Yossi Gil and Tali Shragai discusses the cases where a Java program is dependent on subclasses being constructed under the Uniform Identity inheritance model. It discusses the three key stages of object creation and how each of these contributes to the issues surrounding the construction of objects within class hierarchies. These stages are:

1. Memory allocation
2. Preliminary field initialisation
3. Establishment of invariants

Each of these is dealt with differently across different programming languages. As an example, preliminary field initialisation is approached quite differently in c++ when compared with Java. Java takes the approach of initialising these fields to default values (nulls, zeros and falses) whereas, in the interest of performance, c++ simply leaves these fields with whatever bytes were already present in the memory locations.
Variations between different languages implementations of the final stage, the establishment of invariants, are what leads to differing rules surrounding what the developer can and cant do safely in an object constructor. This is where we find that maintaining a Uniform Identity throughout construction is vital in ensuring that any references to the self which were stored externally during construction remain valid after this process is completed.
We also run into another issue with the changing of the self reference during the construction of an object. During the initialisation of a subclass, it is necessary at some point to initialise the superclass so that its fields are defined after construction. If, during the initialisation of the superclass, the self reference is different to that of the subclass, then any calls to overridden methods will execute the superclass's implementation rather than the subclasss.

2.2 Understanding the Shape of Java Software [1]

This paper details an empirical study of a Java corpus to find details about the structure of typical Java programs. The study collected a large set of Java classes and looked at the frequency of occurrence of a variety of common patterns including the ways developers are typically making use of inheritance and composition. As a result of the study it was found that the frequency of several of these patterns exhibited a power-law.
A further interesting finding of the study was a fairly wide variation in the occurrences of some patterns from project to project. This indicates that some architectural decisions may

contribute heavily to the patterns employed by developers as the project goes on. This also makes it evident that it will be important, in my own empirical study, to ensure that I have a wide range of Java projects from which to gather my statistics to minimise the bias that would be introduced in a smaller dataset.

2.3 Micro Patterns in Java Code [2]

This paper explores the use of micropatterns found in Java programs. These micropatterns are described as similar to design patterns, except standing at a lower, closer to the implementation, level of abstraction.
The patterns I intend to search for as possible examples of forwarding and delegation fit under this definition as each can be expressed as a function on the content of the class.

2.4 The Qualitas Corpus: A Curated Collection of Java Code for Empirical Studies [5]

This paper discusses many of the choices behind the construction of the Qualitas Corpus. Notably, the Java language was chosen for a few specific reasons:

- Open source Java code is abundant and easy to find. Much more so than c#, and similarly to c++.
- Java code tends to be relatively easier to analyse than many other languages including c++ due to some of the limitations it applies to program structure.

2.5 What Programmers Do with Inheritance in Java [7]

This paper goes into detail about the use of inheritance in Java projects and the extent to which classes extend others. To aid with this, the paper also contains a formal definitions of a few terms which are relevant to my study, including subtypes, supertypes and downcalls. These definitions are then used to indicate rates of presence in the Qualitas Corpus of a variety of combinations of the patterns. This is achieved by representing the dependencies within the projects as a graph structure and investigating the properties of that graph.

2.6 How Do Java Programs Use Inheritance? An Empirical Study of Inheritance in Java Software [6]

2.7 Object Inheritance Without Classes [4]

Tim's paper talks about several different models of object based inheritance and the inherent limitations of both. From this, it becomes evident which Java programs are dependent on the Uniform Identity model which is used to construct instances of Java classes within inheritance hierarchies. Knowing which classes are dependent on this model also informs us about which classes would need to be substantially rethought in order to reimplement them in a language which does not support uniform identity.

Chapter 3

Conclusions

The conclusions are presented in this Chapter.

Bibliography

[1] Baxter, G., Frean, M., Noble, J., Rickerby, M., Smith, H., Visser, M., Melton, H., and Tempero, E. Understanding the Shape of Java Software. *Proceedings of the 21st annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications* (2006), 397–412.

[2] Gill, J. Y., and Maman, I. Micro Patterns in Java Code. *Proceedings of the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications* (2005), 97–116.

[3] Gill, J. Y., and Shragai, T. Are We Ready for a Safer Construction Environment? *Genoa Proceedings of the 23rd European Conference on ECOOP* (2009), 495–519.

[4] Jones, T., Homer, M., Noble, J., and Bruce, K. Object Inheritance Without Classes. *30th European Conference on Object-Oriented Programming* (2016).

[5] Tempero, E., Anslow, C., Dietrich, J., Han, T., Li, J., Lumpe, M., Melton, H., and Noble, J. The Qualitas Corpus: A Curated Collection of Java Code for Empirical Studies. *Asia Pacific Software Engineering Conference* (2010), 336–345.

[6] Tempero, E., Noble, J., and Melton, H. *ECOOP 2008 – Object-Oriented Programming: 22nd European Conference Paphos, Cyprus, July 7-11, 2008 Proceedings*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, ch. How Do Java Programs Use Inheritance? An Empirical Study of Inheritance in Java Software, pp. 667–691.

[7] Tempero, E., Yang, H. Y., and Noble, J. *ECOOP 2013 – Object-Oriented Programming: 27th European Conference, Montpellier, France, July 1-5, 2013. Proceedings*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, ch. What Programmers Do with Inheritance in Java, pp. 577–601.