Simulation of Multiple Inheritance(MI) in Java

Sonal Swaroop

College of Basic and Applied Sciences , Middle Tennessee State University 37132

Abstract

Multiple inheritance is a cornerstone of OOPs paradigms with benefits such as reusability (using methods of parent class by child classes), extensibility (extending the parent class logic as per business logic of the child class), data hiding (base class may keep some data private so that it cannot be altered by the derived class), and lesser compilation time. But there is no consensus among researchers on the semantics of multiple inheritance in the presence of method overriding and potential conflicts due to multiple definitions. C++, Common lisp and few other languages supports multiple inheritance while java doesn't support it. It is just to remove ambiguity, because multiple inheritance can cause ambiguity in few scenarios such as Diamond problem. James Gosling quotes *"JAVA omits many rarely used, poorly understood, confusing features of C++ that in our experience bring more grief than benefit. This primarily consists of operator overloading and* ***multiple inheritance***. Java creators provided users with much easier and robust alternative to multiple inheritance i.e. interfaces, composition and abstract classes that alleviate problem during casting and constructor chaining. The primary objective of this research study is to develop a trade-off between "multiple inheritance in C++" and "substitution for multiple inheritance in Java" and conclude if multiple inheritance is good programming technique and if Java should introduce direct implementation of multiple inheritance. The secondary objective is to introduce novel ways to substitute multiple inheritance in Java such as "twin pattern" and "dynamic multiple inheritance", and conclude if such implementations may bring much easier and feasible alternatives to "interfaces" .

Background

•Diamond Problem has become one of the major motivations for developers to create a language without the evils of MI such as Java.

•Java is the first language that has separated polymorphism and inheritance by providing interfaces which represents a subtyping relationship, whereas extending a class represents the more traditional combination of subtyping and behavior inheritance.

•Java has various techniques such as abstract classes, compositions, default methods etc other then interfaces to substitute for MI.

Methods

•In order to develop a tradeoff between direct implementation in C++ and substitution of MI in Java ,a comparison table given as table1 will be derived

•To develop a tradeoff, two sets of programmers has been created , with first .group having 2 developers, each affluent in basic C++ and Java programming and second group. having one exclusive Java programmer and one exclusive C++ programmer.

•A set of MI related programming problems with 7-days time, has been handed over to the first group of programmers.

•Novel ways to introduce MI into Java : **Twin Pattern** can be used to simulate MI in a language that does not support this feature. It also avoid certain problems of multiple inheritance such as name clashes.

. Novel ways to introduce MIthe program that create into Java : Java enables them, so primary goal here dynamic combination of is to infuse dynamic MI in classes which is complex Java in such a way that it is and unsafe, and the safe in large sets and resulting dynamic classes useful. are poorly integrated into

Figure 1:MI in C++

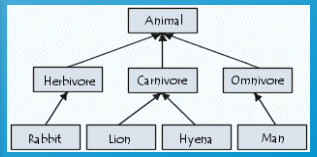
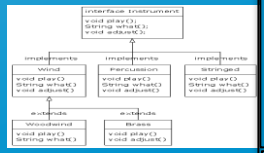


Figure 2 :MI in Java



Results

1)Trade off between C++ and Java: A set of MI related programming problems with 7-days time, has been handed over to the first group of programmers.

Table 1 : Comparison over MI

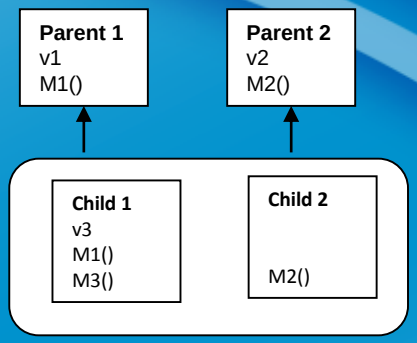
Metrics / Platforms	Java	C++
Metrics of understandability		
Metric of modifiability		
DIT		
NOC		
Simplicity of Tree Structure		
User Efficiency		
Control over code		
Error rate		

2)Twin Pattern in Java :Every child class is responsible for the protocol inherited from its parent. It handles messages from this protocol and forwards other messages to its partner class.

•Subclassing the Twin pattern: If the twin pattern should again be subclassed, it is often sufficient to subclass just one of the partners, for example Child1. This solution has the problem that Sub is only compatible with Child1 but not with Child2.

•More than two parent classes. The Twin pattern can be extended to more than two parent classes in a straightforward way. For every parent class there must be a child class. All child classes have to be mutually linked via fields. It is rare that a class inherits from more than two parent classes.

Figure 3: Twin pattern structure



3) With dynamic inheritance, we create a composite object with the separate objects wrapped together according to their priority. For the outside world, they can call any of the interface functions on the composite object. Dynamic inheritance is useful when there are lots of different implementations possible for each of the interface functions. This avoids creation of large number of class definitions with all the possible combination of interface functions.

References

1) Ernst, Erik. 2002. "Safe Dynamic Multiple Inheritance," Journal

2)Thirunarayan, Krishnaprasad, Günter Kniesel, and HaripriyanHampapuram. 1999. "Simulating Multiple Inheritance and Generics in Java." Computer Languages 25 (4): 189–210.1-22

3) Meyer B.1997. Object-oriented software construction. 2nd ed. Englewood Cli?s, NJ: Prentice-Hall

4) Sakkinen M. 1992. Inheritance and Other Main Principles of C++ and Other Object Oriented Languages. PhD thesis, University of Jyvaskyla.

5) Tempero E. and Biddle R. 2000. Simulating Multiple Inheritance in Java. Journal of Information and Software Technology, (55):87–100.

6) Muhlhauser, Max,Gurevych, Iryna. 2008. Handbook of Research on Ubiquitous Computing Technology for Real Time Enterprises. 62-67

7) Adriana B. Compagnoni and Benjamin C. Pierce. 1996. Higher order intersection types and multiple inheritance. Mathematical Structures in Computer Science, 6(5):469–501,

8) Mössenböck, Hanspeter. 2000. "Twin — A Design Pattern for Modeling Multiple Inheritance." Perspectives of System Informatics 1755: 358–69. doi:10.1007/3-540-46562-6_31.

9) Gosling ,James and McGilton ,Henry. 1996. The Java Language Environment," (Sun Microsystems)

10) Meyers ,Scott. (2nd Edition) (Addison-Wesley Professional Computing) Effective C++: 50 Specific Ways to Improve Your Programs and Design

11) Sutter,H. 1998. Uses and Abuses of Inheritance, Part 2(C++ Report, 10(9))