Interfaces with Skeletal Implementations in Java

Markus Mohnen

Lehrstuhl für Informatik II, RWTH Aachen, Germany
mohnen@informatik.rwth-aachen.de

Extended Abstract

Java's interfaces allow the definition of class properties independently of class inheritance. Since interfaces allow multiple inheritance, their potential for reusable code is higher than those of abstract classes. We present a technique for augmenting interfaces with skeletal implementations by using Java's inner classes. Using this technique, we propose an extension of Java for providing skeletal implementations in interfaces.

Multiple inheritance of method implementations from more than one superclass in object oriented languages has been identified as a cause of problems for a long time (see for instance [Sak92b,Sak92a]). One of the main problems is that it seems impossible to define a satisfactory general strategy for choosing or combining inherited method implementations.

Newer languages like Ada95 [DT97] or Java [GJS96,AG96] avoid these problems by not allowing multiple inheritance of method implementations at all. In contrast to Ada95, however, which does not support multiple inheritance as a language construct, Java allows multiple inheritance of *signatures*: Interfaces allow the definition of properties which can be implemented by classes[1]. Since interfaces can be used just like classes in declarations and signatures, it is possible to base programs on *properties of classes* instead of classes. Furthermore, the inheritance hierarchy of interfaces is independent of the class inheritance tree. Therfore, this language feature gives a higher potential to produce reusable code than *abstract classes*, i.e. classes where the implementation of some methods is omitted. The Java 2 API makes extensive use of interfaces. For instance, the package `java.util` contains six interfaces hierarchies.

In many cases it would be useful to equip interfaces with *skeletal implementations* of methods, which can be used by a class implementing the interface. Skeletal implementations are useful since they reduce the effort required to implement the interface. They are especially interesting if there is a canonical way to implement methods of the interface in terms of some other methods. In these cases, an implementation can be obtained by implementing the base methods and use skeletal implementations of the other methods. The usefulness of skeletal implementations has also been seen by the Java developers. In the Java 2 API there are abstract classes which provide skeletal implementations for some of the interfaces. For instance, the class `java.util.AbstractCollection` is an abstract class implementing `java.util.Collection` except for the methods `iterator` and `size`.

The approach of adding abstract classes containing skeletal implementations has drawbacks for the implementation of the interface and for the implementation of the abstract classes:

- To use the skeletal implementation of the interface, a class must extend the abstract class. Consequently, the programmer is no longer free to create an inheritance hierarchy matching the needs of the problem at hand. Altogether, this abolishes the advantages of interfaces over abstract classes.
- To provide skeletal implementations for all interfaces in an interface hierarchy with multiple inheritance, it is unavoidable to duplicate code in the abstract classes.

We suggest a different approach for providing skeletal implementations of interfaces. Java's inner classes can be used to add default implementations to an interface without creating a separate abstract class. Using this approach, we avoid the problems mentioned in the last paragraph. Based on this technique, we propose an extension of Java, which allows to insert skeletal implementations directly in interfaces.

References

[AG96] K. Arnold and J. Gosling. *The Java Programming Language*. Addison-Wesley, 1996.

[DT97] R. A. Duff and T. S. Tucker, editors. *Ada 95 Reference manual*. Number 1246 in Lecture Notes in Computer Science. Springer–Verlag, 1997.

[GJS96] J. Gosling, B. Joy, and G. Steele. *The Java Language Specification*. The Java Series. Addison Wesley, 1996.

[HPW92] P. Hudak, S. L. Peyton Jones, and P. Wadler *et. al.* Report on the Programming Language Haskell — A Non-strict, Purely Functional Language. *ACM SIGPLAN Notices*, 27(4), 1992.

[Sak92a] M. Sakkinen. A Critique of the Inheritance Principles of C++. *Computing Systems*, 5(1):69–110, 1992.

[Sak92b] M. Sakkinen. The Darker Side of C++ Revisited. *Structured Programming*, 13(4):155–177, 1992.

[1] Interfaces are similar to what is called type classes in the functional language Haskell [HPW92].