Summary of Method for Renaming Identifiers of a Computer Program

March 7, 2002

by
Balamurugan Chirtsabesan

1 Introduction

The US Patent by Paul Tyma proposes a method for renaming the identifiers in a Java program. The key to writing a good program is to use meaningful names for identifiers. In Java, an identifier could refer to the name of a class, a method or a variable. Given that Java applications can use many classes, the storage required for storing their names can be significant compared to the total size of the class files. Also the practice of using long, meaningful identifiers makes the Java programs more comprehensible and thus simplifies reverse-engineering. Tyma proposes a method for renaming identifiers with smaller names so that they reduce the size of the class files and also obfuscate the program. Local variables within a method are not renamed as they are not stored in the class files after compilation.

The paper describes a technique for optimally renaming programs to an optimal set of identifiers. This means that redundancy is introduced by renaming many of the identifiers to the same string. Introducing redundancy into renaming also has the effects of reducing the size of the constant pool and obfuscating the program. Thus a minimal set of new names are used in renaming. These could be simple alphabetic letters like a,b,c,... or a minimal set of new names could be chosen by the implementor or user.

The paper also describes a way in which we could de-obfuscate the obfuscated code. For this purpose, redundantly named identifiers are renamed

uniquely. By removing such redundancy, the Java code will become easier to understand and maintain.

2 Renaming Class Names

In Java, class names have two possible parts, a package name and the actual class name. Although separate at compile time, the package name becomes part of the class name after compilation. Therefore package and class names are treated as a single unit and all package organization aspects are disregarded.

The original class names are replaced sequentially with new class names such as a,b,c,... or short strings like red,orange,yellow,...

3 Renaming Field Names

Field names are not affected by inheritance hierarchies. They can be distinguished by their name and class membership. As a result, we only need to distinguish field names within a given class. In one method, all the field names of all classes are replaced sequentially with new names like a,b,c,... In another method, the renaming restarts with 'a' for each class. Thus naming redundancy is introduced. If we are intersted in de-obfuscation, then non-redundant unique names are always proposed for renaming.

Note that in the methods proposed by Tyma, local variable names are not renamed since they do not exist after compilation.

4 Renaming Method Names

Methods are special in Java since along with the method name, the parameter list of the method uniquely identifies it within a given class. This allows us to achieve optimal renaming. This means that different methods within the same class can be renamed to the same new name as long as their parameter lists are different. One needs to be careful while renaming the methods of a class because unlike field names, method names are affected by inheritance hierarchies as we can have inherited and overridden methods. So we have to make sure that while renaming a method that overrides another method from its parent class, we rename both the overriding and overridden method to

the same name. Hence the process of method renaming involves identifying naming dependencies by building the class inheritance hierarchy tree.

Building the inheritance hierarchy tree requires us to identify the interfaces implemented by various classes as well as looking at the access modifiers of the methods, which specify whether the method can be inherited or not. For the purpose of renaming, interfaces are treated the same as Java classes and affect the hierarchy tree accordingly. Also in Tyma's algorithm for renaming methods, static methods are treated the same as private methods and hence are never inherited.

The paper describes a systematic procedure for renaming the methods of a Java program which is outlined as follows :

1. Generate the class inheritance hierarchy tree and identify naming dependencies.

2. Identify all the leaf classes in the hierarchy tree.

3. Method lists are generated for each leaf class which include all methods that each leaf class inherits as well as methods that are included within each class. Access modifiers are used in this determination.

4. All incidences of superclasses that contain default methods are noted. If any of these superclasses are of different packages than any of its subclasses, then the default methods of the superclass are added to the method lists of the subclass.

5. Renaming of method names is performed with the help of a Dependency Subroutine that is called first with the methods that are listed in the method lists and then with the remaining methods.

5 Dependency Subroutine

The Dependency Subroutine is called with an input parameter X being the method name of the method to be renamed. If X has not already been assigned a new name, a new name is proposed for X from an ordered list of names. By starting from the beginning of the list each time, we can introduce redundancy into the renaming as well as provide optimal renaming. If the goal of renaming is to de-obfuscate a program, then we just traverse the

ordered list of new names without starting at a pre-determined location each time.

Once a new name is proposed, all method lists are scanned where X is a member. If another method with the same signature as X is found to already have the proposed name, then a new name has to be proposed for X. Once a name that can be assigned to X is found, all method lists are scanned and all the overridden versions of X are identified. The Dependency Subroutine is called recursively to assign these overridden methods the same new name as X thus preserving naming dependencies.

6 Computer System

The paper also describes a simple computer system in which software for performing the various methodologies described above is embedded. This is essential (or was essential at the time) for patenting any algorithm such as that described above.