Java 6 Decompiler

Joshua Cranmer
TJHSST Computer Systems Lab
2007-2008

Abstract

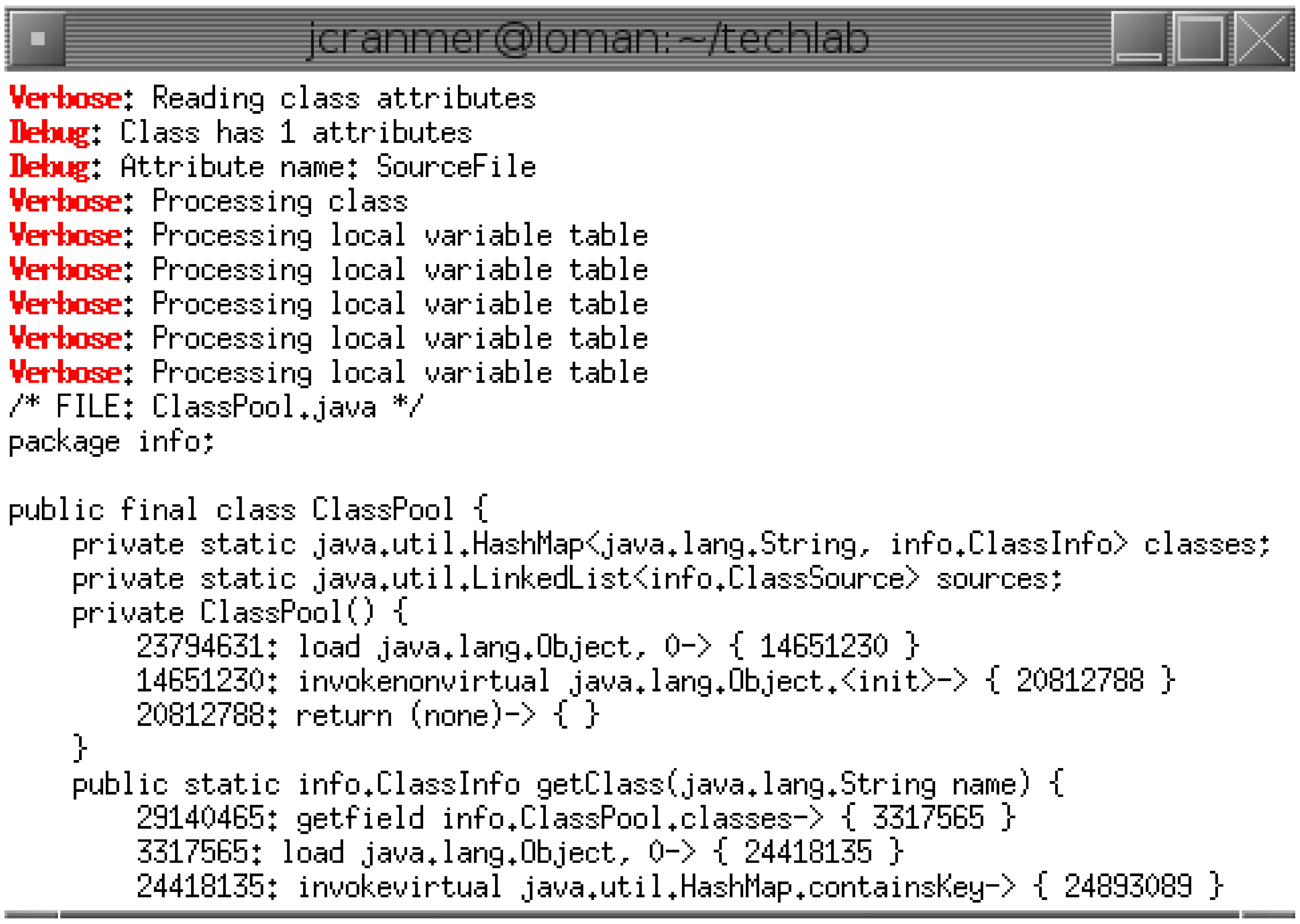
This project aims to create a decompiler capable of processing outputted Java 6 bytecode into fully-recompilable and functionally-equivalent source code.

Reasons for Decompilation

•Finding bugs in program
•Finding vulnerabilities
•Finding malware
•Compiler code verification
•Comprehending algorithms
•Creating interoperability
•Induce customizability
•Porting code
•Create maintainable source code
•Fixing bugs without patching binaries
•Add features to a program

Procedures and Methods

The decompiler works in a multi-phased approach. First, the class file is fully parsed and stored in memory. Then, the code execution bodies are processed through several transformation filters until readable source code is produced. Next, various filters are applied to make the source code more readable. Finally, everything is fully decoded and then printed out into class files.



```
Verbose: Reading class attributes
Debug: Class has 1 attributes
Debug: Attribute name: SourceFile
Verbose: Processing class
Verbose: Processing local variable table
Verbose: Processing local variable table
Verbose: Processing local variable table
Verbose: Processing local variable table
Verbose: Processing local variable table
/* FILE: ClassPool.java */
package info;

public final class ClassPool {
    private static java.util.HashMap<java.lang.String, info.ClassInfo> classes;
    private static java.util.LinkedList<info.ClassSource> sources;
    private ClassPool() {
        23794631: load java.lang.Object, 0-> { 14651230 }
        14651230: invokenonvirtual java.lang.Object.<init>-> { 20812788 }
        20812788: return (none)-> { }
    }
    public static info.ClassInfo getClass(java.lang.String name) {
        29140465: getfield info.ClassPool.classes-> { 3317565 }
        3317565: load java.lang.Object, 0-> { 24418135 }
        24418135: invokevirtual java.util.HashMap.containsKey-> { 24893089 }
```

Example screenshot of running code. Note the use of proper indentation and (not seen here) proper 80-character overflow.

The output inside the blocks is a dump of the internal code graph.