# 자바 클래스 파일에 대한 시각화 실행 분석기

고 광 만†


## 요 약

자바 언어는 빠른 속도로 인터넷 및 분산 응용 분야 등에서 활용되고 있으며 단순히 응용 소프트웨어를 개발할 수 있는 프로그래밍 언어 이상으로 활용 범위가 확대되고 있다. 특히, 실행 환경인 자바 가상 기계에 연관되어 다양한 연구가 진행되고 있으며 자바 클래스 파일에 대한 분석 및 응용 분야에 적합한 형태로 정보를 활용할 수 있는 다양한 시도가 진행되고 있다. 자바 언어에 대한 클래스 파일은 소스 프로그램의 의미를 자바 가상 기계에서 실행가능한 형식으로 변환된 형태이다. 이러한 클래스 파일의 구조 및 실질적인 실행 과정에 대한 분석은 디컴파일러 구성, 소스 프로그램의 디버깅 등에 편리성을 지원할 수 있다. 본 논문에서는 이러한 클래스 파일에 대한 분석을 비롯하여 실제로 실행되는 과정을 보다 시각적으로 표현하기 위한 실행 분석기 개발에 관한 연구이다. 이를 위해 클래스 파일의 내용을 GUI 환경에서와 같이 접근 및 표현이 용이하도록 구현하였다. 클래스 파일이 포함하고 있는 정보들을 Constant_Pool 부분, Class_file 부분, Interface 부분, Field 부분, Method 부분, Attribute 부분으로 나뉘어서 나타내도록 해주었다. 또한 클래스 파일의 실행 과정에서 핵심 정보를 저장하고 있는 메소드 영역 정보, 오퍼란드 스택 정보, 지역 변수의 정보를 시각적으로 표현하였다.


# Visualized Execution Analyzer for the Java Class File

Kwang Man Ko†


## ABSTRACT

The Java language is rapidly being adopted in the Internet. The distributed applications and their application range are being expanded beyond just a programing language and developed into software applications. A variety of researches are going on with regard to the Java Virtual Machine runtime environment and methods of analyzing the Java class files and utilizing the information for applications. A class file is a converted file that is executable by the Java virtual machine. Analysis on the class file structure and the runtime processes will be convenient in arranging the decompilers and debugging the source programs. This paper is about the runtime process analyzer that presents the runtime processes, including class files, more visually. The content of a class file will be easily accessed and expressed as in a graphic user interface. The information in the class file displayed is divided into Constant_Pool, Class_file, Interface, Field, Method and Attribute with information on method area, operand stack and local variables expressed visually.


**키워드 : 자바 클래스 파일(Java Class File), 브라우저(Browser)**

## 1. 서 론

자바 프로그래밍 언어는 인터넷 및 분산 환경 시스템에서 효과적으로 응용 프로그램을 작성할 수 있도록 설계된 언어로서 객체지향 패러다임의 특성 및 다양한 개발 환경을 지원하고 있다. 자바 언어는 특정한 하드웨어, 운영체제에 영향을 받지 않고 동작할 수 있는 높은 이식성을 가지고 있다[1, 2].

객체지향 언어로 작성된 프로그램은 프로그램의 수행에 필요한 많은 정보가 숨겨져 있다. 그 이유는 여러 개의 객체들로 구성된 전체 프로그램 중에서 일부 객체만을 사용자가 실제 작성하고 대부분의 객체는 시스템 혹은 타인이 작성한 객체를 재사용하기 때문이다. 이러한 이유로 객체지향 프로그램을 분석하기가 쉽지 않다. 자바 프로그램의 경우도 마찬가지이다. 그러나 자바의 경우는 컴파일 과정을 통해 생성된 클래스 파일에 프로그램의 수행과 관련된 정보가 숨겨져 있게 된다. 즉, 자바 언어 시스템에서는 응용 프로그램을 플랫폼에 독립적으로 실행시키기 위해 자바 가상 기계에서 클래스 파일을 사용하고 있다. 따라서 클래스 파일은 원시 프로그램에 대한 모든 정보를 갖고 있으며 이러한 정보를 효과적으로 저장하고 활용할 수 있는 다양한 시도가 많이 진행되어 왔다. 특히, 생성된 클래스 파일 정보에 대한 분석을 통해서 클래스 파일의 최적화와 같은 보다 개선된 실행 방법이 제시되어 왔다.

※ This work was done as a part of Information & Communication Fundamental Technology Research Program supported by Ministry of Information & Communication in Republic of Korea.
† 종신회원 : 상지대학교 컴퓨터정보공학부 교수
논문접수 : 2004년 5월 23일, 심사완료 : 2004년 9월 9일

자바 클래스 파일은 바이너리 형태로서 자바 가상 기계에서 실행되어 실행 결과를 생성한다. 이러한 클래스 파일의 구조는 실행에 필요한 모든 정보를 저장하고 있으므로 그 내용 및 구조를 이해하는데 많은 기초적인 지식을 필요로 한다.

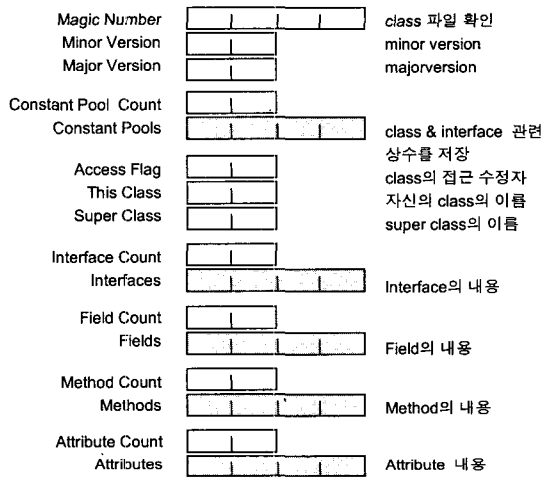
바이너리 형태로 구성된 클래스 파일은 자바 가상 기계에서 올바른 수행을 위한 많은 정보를 가지고 있으며 실질적으로 클래스 파일의 구조 및 정보를 추출하기 위한 도구들이 사용되고 있다. 하지만 이러한 도구들은 대부분 텍스트 형식으로 구성되어 클래스 파일의 내용을 접근하는데 많은 불편함을 가지고 있다. 또한 클래스 파일의 핵심 부분인 바이트 코드를 이해하기 위해서는 복잡한 클래스 파일에 대한 접근이 제한되고 있다. 따라서 본 논문에서는 클래스 파일의 정보를 보다 편리하게 접근할 수 있도록 클래스 파일의 구조를 6개의 핵심 부분으로 나누어 재구성하였으며 각각의 정보를 시각화 하여 화면 좌측에 표현하였다. 그리고 자바 가상 기계에서는 니모닉 코드들을 분석하여 적절한 수행을 통해 실행하게 되는데 실제로 자바 가상 기계의 내부에서 니모닉 코드들이 수행되는 과정을 보다 편리하게 이해할 수 있도록 본 논문에서 자바 클래스 파일을 분석하여 클래스 파일의 내용 및 자바 가상 기계에서 실제로 수행되는 과정을 보다 시각적인 형식으로 구현하였다. 이를 위해서 자바 가상 기계에서 명령어들을 포함하고 있는 메소드를 6개의 핵심 부분으로 나누어 분석한 후 각각의 니모닉 코드들이 자바 가상 기계에서 수행되는 과정을 보다 편리하게 참조할 수 있도록 하였다.

## 2. 기반 연구

### 2.1 클래스 파일 구조

자바 클래스 파일은 자바 프로그램을 위해 정의된 바이너리 형식의 데이터를 의미하게 된다. 파일 이름과 클래스 이름은 같게 되며, 자바 소스 파일 안에 있는 클래스의 개수만큼 생성되게 된다. 즉 소스 파일 안에 아무리 많은 클래스가 정의되어 있더라도 생성되는 클래스 파일은 정의된 클래스 파일당 하나씩 존재하게 되는 것이다. 클래스 파일은 8비트 바이트들의 이진 흐름으로 구성되며 데이터 아이템들은 순차적으로 클래스 파일에 저장이 되게 된다. 모든 16비트와 32비트 크기들은 2개 혹은 4개의 8비트 바이트 각각 읽음으로써 구성되며, 1바이트 이상 차지하는 몇 개의 연속된 바이트에 높은 바이트를 먼저 써 넣는 방식(big-endian)으로 만들어지게 되며 (그림 1)과 같은 구조를 가지고 있다[1, 2].

모든 클래스 파일의 처음 4바이트는 magic number로써 0XCAFEBABE라는 자바 클래스 파일임을 나타내주는 고유의 값을 갖게 된다. 즉, 파일이 0XCAFEBABE로 시작하지 않으면 그 파일은 자바 클래스 파일이 아니게 된다. minor_version과 major_version은 클래스 파일을 생성한 컴파일러의 버전을 나타내는 것으로써 JDK 2 버전부터는 minor_ version은 0, major_version은 46 값을 저장하고 있다. constant_pool은 클래스나 인터페이스에서 선언 또는 정의된 상수 값을 저장하고 있는 메모리 공간으로서 리터럴 문자열, final 변수, 클래스 이름, 메소드 이름 등을 포함하고 있다. 또한 constant_pool_count는 constant_pool에 들어 있는 모든 엔트리의 개수를 나타내며 constant_pool 테이블의 처음에 나오는 1바이트가 어떤 종류의 상수인지를 결정하게 된다. access flag는 클래스의 수정자를 비트 마스크 값으로 나타낸 것이고 this class 및 super class는 자신 클래스의 이름과 슈퍼 클래스 이름을 각각 나타낸다. Object 클래스가 모든 클래스의 부모가 되기 때문에 프로그램에서 슈퍼 클래스를 지정하지 않았다면 java.lang.Object가 슈퍼 클래스가 된다. interface, field, method, attribute는 각각의 정보를 가지게 된다.




(그림 1) 클래스 파일 구조

### 2.2 클래스 파일 분석 및 실행 과정 분석 도구

썬 마이크로 시스템즈에서 제공되는 JDK[1, 2]에서는 javap -c 옵션을 이용하여 클래스 파일의 내부 구조를 접근할 수 있다. 클래스 파일의 정보는 메소드 부분에 대한 Opcode 부분으로서 JVM에서 수행되는 역할을 상세하게 볼 수 있다.

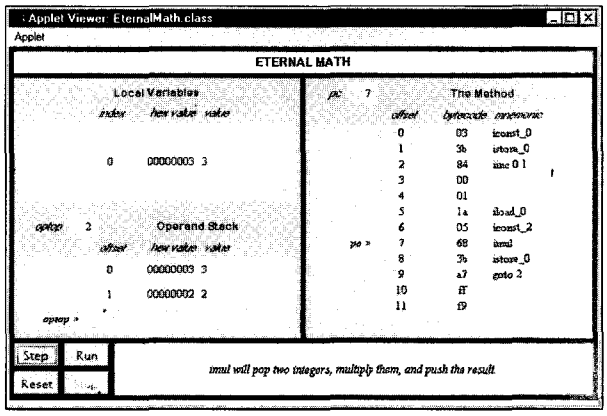
DJ Java Decompiler[19]는 시각화 브라우저를 사용해 클래스 파일 정보 표현 및 역컴파일(decompiler)까지 가능한 도구이다. 하지만 jad[20]를 이용하여 클래스 파일을 분석하였기 때문에 JVM에서 지원하는 javap의 기능과 유사하여 클래스 파일의 내용 전체를 브여주지 못하고 JVM에서 실행되는 메소드를 중심으로 Opcode에 대한 정보를 보여준다.

DumpClass[12]는 Oolong의 Dumpclass로서 클래스 파일에 대한 모든 정보를 알 수 있는 도구이지만 텍스트 형태로 표시되기 때문에 일반 사용자가 클래스 파일에 대한 구체적인 지식 없이는 접근하기 쉽지 않은 도구이다.

ClassCracker[21]는 Mayon Software Research의 클래스 파일 분석 도구로서 java로 작성되어 클래스 파일을 덤프시

켜 내용을 볼 수 있도록 한 도구이다. 하지만 전체 클래스 파일에 대한 내용을 텍스트 형식과 같이 동시에 표현하므로서 클래스 파일에 대한 접근이 쉽지 않다

Artima Software[18]사의 Eternal Math는 자바 가상 기계가 소수의 바이트 코드 명령어를 시뮬레이션하는 과정을 애플릿 뷰어나 웹 브라우저를 통해 보여주는 자바 애플릿이다. 이 시뮬레이터는 가상 기계 코드 입력을 파일이 아닌 프로그램내의 초기화 문장으로 처리하고 있으며 특정 클래스의 한 메소드내의 가상 기계 코드가 실행되는 과정만을 보여준다. 버튼에 대한 마우스의 이벤트를 통해서 메소드의 실행 순서, 오퍼랜드 스택 및 지역 변수들의 변화를 (그림 2)와 같이 보여준다.



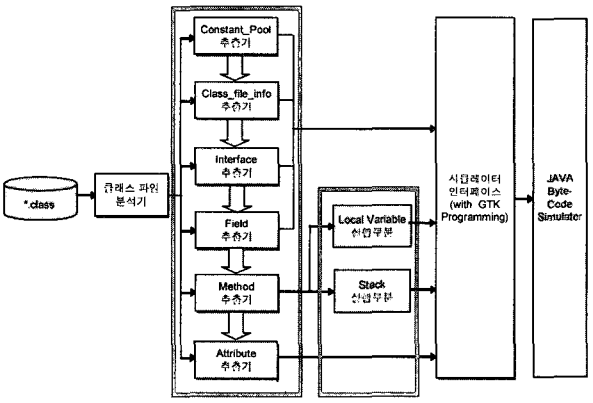

(그림 2) Eternal Math의 바이트코드 시뮬레이션

김도우, 정민수[13]는 자바 바이트코드 실행 시각화 도구의 개발을 통해 자바 프로그램을 분석하고 명확한 이해, 자바 가상 기계의 동작 과정 분석을 통한 컴파일러의 최적화 혹은 프로그램의 최적화 활용이 가능하도록 하였다. 또한 사용자가 작성한 클래스 파일 뿐만 아니라 시스템 혹은 타인이 제공한 클래스 파일내의 바이트코드도 시각적으로 시뮬레이션하면서 실행시킬 수 있어 다른 사람이 작성한 프로그램도 쉽게 이해할 수 있도록 하였다. 자바 가상 기계의 구성 요소들을 시각화하여 바이트코드가 실제 수행되는 과정을 보여주는 자바 바이트코드 시뮬레이터는 바이트코드 분석부, 메소드 실행부, 스택 프레임 실행부로 크게 구성되어 있다. 바이트코드 분석부는 입력된 클래스 파일을 분석하여 클래스 파일 내부에 저장되어 있는 상수 풀 정보, 클래스 이름, 클래스 내부의 메소드, 지역 변수, 각 메소드를 구성하는 바이트코드, 각 메소드가 필요로 하는 스택의 크기 및 지역 변수의 크기 등의 정보를 추출하여 내부 자료 구조인 methodPoolInfo에 저장한 후 다음 단계인 메소드 실행부에 의해 참조될 수 있도록 하였다. 메소드 실행부에서는 methodPoolInfo 정보를 참조하여 각각의 정보를 사용자 인터페이스를 통해 출력하였다. 스택 프레임 실행부는 메소드 내의 실제 코드의 실행으로 인해 발생되는 스택 연산의 결과 반영으로 초래되는 오퍼란드 스택의 변화 및 지역 변수의 변화를 사용자 인터페이스 구성 요소인 오퍼란드 스택 영역, 지역 변수부를 통해 보여주는 역할을 한다. 하지만 자바 가상 기계의 실행 과정 중 적재, 배치, 초기화 동작이 가상 기계에 의해 선행되었다는 가정하에 실행되는 클래스 파일의 분석을 통해 분석된 내용들을 기반으로 시뮬레이션하기 때문에 문제점을 가지고 있다.

## 3. 시각화 실행 분석기

### 3.1 개 요

자바 언어의 클래스 파일은 소스 프로그램의 의미를 자바 가상 기계에서 실행가능한 형태이다. 이러한 클래스 파일의 구조 및 실질적인 실행 과정에 대한 분석은 디컴파일러 구성, 소스 프로그램의 디버깅 등에 편리성을 지원할 수 있다. 본 논문에서는 이러한 클래스 파일에 대한 분석을 비롯하여 실제로 실행되는 과정을 보다 시각적으로 표현하기 위한 실행 과정 분석기를 설계하고 구현한다. 이를 위해 클래스 파일의 내용을 GUI 환경에서와 같이 접근 및 표현이 용이하도록 (그림 3)과 같은 과정을 통해 자바 클래스 파일 시각화 브라우저 및 실행 분석기를 구현하였다.




(그림 3) 자바 클래스 파일 실행 분석기 구성

클래스 파일이 포함하고 있는 정보들을 Constant_Pool 부분, Class_file 부분, Interface 부분, Field 부분, Method 부분, Attribute 부분으로 나뉘어서 나타내도록 해주었다. 또한 클래스 파일의 실행 과정에서 핵심 정보를 저장하고 있는 메소드 영역 정보, 오퍼란드 스택 정보, 지역 변수의 정보를 시각적으로 표현하였다. 전체적인 구현 과정은 *.class 파일을 입력으로 받아 Constant_Pool, Class_file_info, Interface, Field, Method, Attribute 정보 분석을 6단계로 나누어서 수행한다. 분석된 내용중 자바 가상 기계상에서 실질적으로 실행되는 동작 과정 분석하고 표현하기 위해서 method 부분을 따로 읽어 들여 local variable 영역 정보와 stack 영역 정보에 어떠한 내용들이 저장되고 변화하는지에 대해 분석

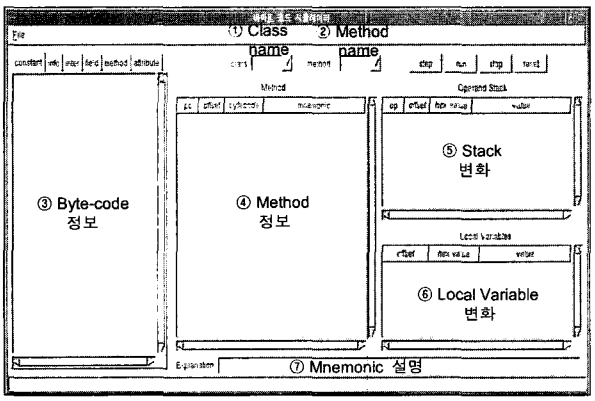
하며 그 과정을 GTK(GIMP Tool Kit)를 이용하여 시각화하여 표현한다.

자바 가상 기계에서 클래스 파일이 실질적으로 수행되는 과정을 표현하기 위해서는 클래스 파일의 분석이 선행되어야 한다. 자바 컴파일러를 통해 생성된 클래스 파일을 구성하는 내용들 간의 상호 연관 관계 및 자바 가상 기계 동작 시에 어떻게 유기적으로 연동되는지 이해할 필요가 있다. 클래스 파일 분석기에서는 입력된 클래스 파일을 분석하여 클래스 파일내부에 저장되어 있는 모든 정보들을 추출하게 된다. 입력된 클래스 파일에 대해 정의된 포맷에 따라 그 내용을 분석한다. 분석된 클래스 파일에서 메소드 부분이 실제 명령어가 사용되고 실행되는 부분이기 때문에 메소드 부분의 내용을 메소드 추출기의 입력으로 전달하게 된다.

메소드 추출기에서는 클래스 파일 분석기를 통해 분석된 클래스 이름, 메소드 이름, 메소드 실제 코드, 메소드 실제 코드에 해당하는 가상 기계의 명령어, 실행될 위치를 나타내는 프로그램 카운터를 자바 가상 기계의 구성 요소들로 구성된 인터페이스를 통해 표현한다. 프로그램 카운터는 명령어의 수행에 의해 발생하는 오퍼란드 스택과 지역 변수 분석부로 전달된다. 오퍼란드 스택과 지역 변수 분석부에서는 메소드 분석부를 통해 분석된 내용중 명령어의 실행에 의해 발생되는 스택 연산의 결과와 지역변수 부분의 결과를 분석하여 출력하여 준다. 이때 각각의 명령어에 따른 의미도 함께 출력하여 주게 된다.

### 3.2 설계 및 구현 환경

본 연구의 수행 결과는 리눅스 환경에서 작동하도록 설계되었으며 인터페이스를 표현하게 위해 GTK(GIMP Tool Kit)를 이용하여 개발하였다. (그림 4)는 클래스 파일 실행 분석기의 전체적인 화면 구성으로서 File 메뉴를 이용하여 클래스 파일을 선택한 후 해당 클래스 파일에 대한 내용을 보여주게 된다.




(그림 4) 클래스 파일 시각화 실행 분석기 메인 화면

① Class name : 입력으로 받은 클래스의 이름을 나타낸다.

② Method Name : 참조하고자 하는 메소드 이름을 선택하는 부분이다.

③ Byte-Code : *.class 파일에 대한 Constant_Pool, lass_fileinformation, Interface, Fields, Methods, Attributes 각각에 대한 정보를 GTK를 이용하여 출력한다.

④ Method : 선택한 메소드에서 사용되는 명령어들이 표현된다. PC는 현재 읽혀지고 있는 카운터의 위치이고, offset는 명령어들의 위치, 바이트 코드는 명령어들이 본래 어떤 바이트 코드로부터 왔는지를 알 수 있게 해주며 니모닉 명령어들을 보여주게 된다.

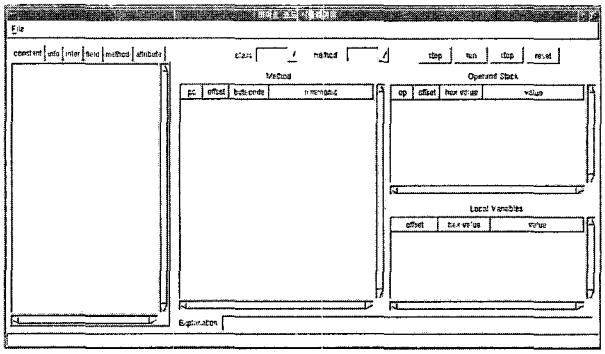
⑤ Stack : 명령어가 수행될 때 스택에서 일어나는 동작 상태를 표시한다.

⑥ Local Variable : 지역변수들의 변화에 대해 보여주게 된다.

⑦ Mnemonic 설명 : 해당 명령어가 수행되면 어떠한 의미를 갖는지에 대해 설명을 나타낸다.

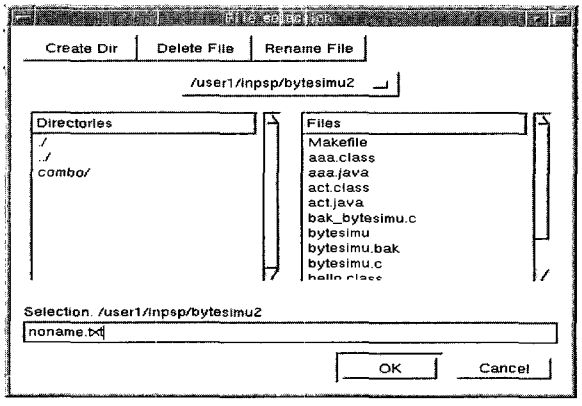
### 3.3 구현 결과

실질적으로 구현된 클래스 파일 실행 분석기는 (그림 5)와 같이 표현되며 File 메뉴를 이용하여 파일 입력과 종료 기능을 수행한다.



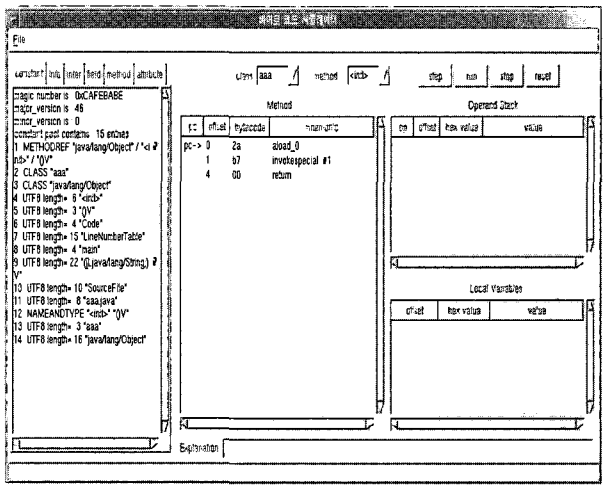
(그림 5) 클래스 파일 시각화 실행 분석기 초기 화면

(그림 6)은 특정 클래스 파일을 선택하는 결과를 나타낸다. 우측의 Files 부분에서 필요한 파일을 선택한다.



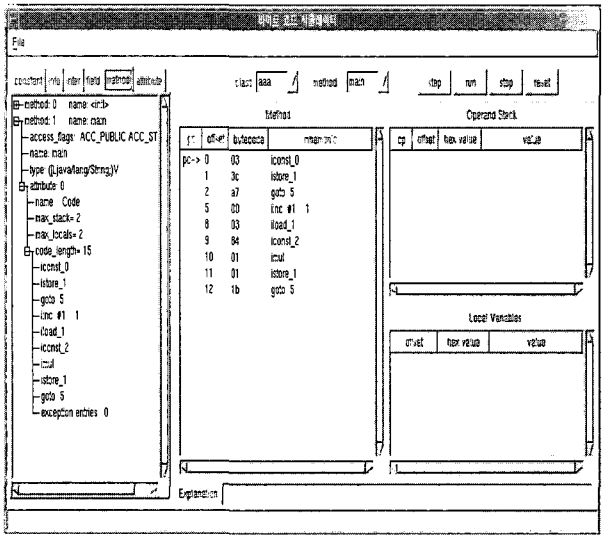

(그림 6) 클래스 파일 선택

특정 클래스 파일(aaa.class)을 입력으로 받아 각각에 대한 정보가 (그림 7)과 같이 표현된다. 좌측 부분에서 aaa.class 파일에 대한 분석 정보가 나타난다. (그림 7)에서는 constant_pool의 내용을 볼 수 있도록 선택하였으며 class 이름, method는 init()라는 method가 나타나게 된다.



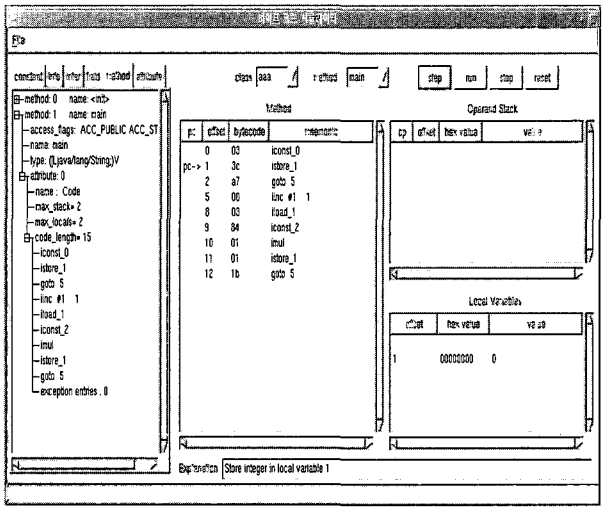
(그림 7) 클래스 파일 정보 표현

(그림 8)은 클래스 파일 정보중 메소드 정보로 표현한 것으로 메소드에 대한 정보를 보다 쉽게 참조할 수 있도록 하기위해 트리 구조를 사용하여 init() 메소드와 main() 메소드 두가지를 트리 형태로 보여준다. 이 부분에서는 메소드에 대한 모든 정보가 들어 있기 때문에 우측에서 메소드의 pc가 변화할 때 어느 부분에 위치하는지 쉽게 확인할 수 있게 된다.



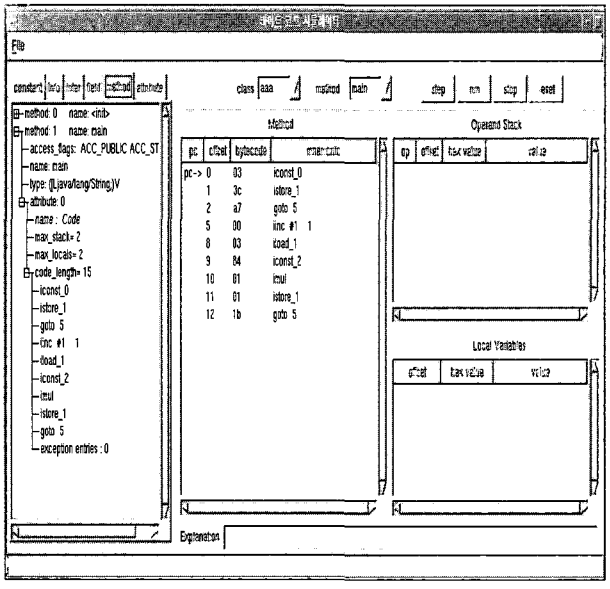
(그림 8) 클래스 파일의 메소드 정보 표현

step 버튼을 누르면 PC가 0에서 1로 증가함에 따라 1 다음 명령어인 istore_1 명령어를 읽어들여 지역 변수의 첫 번째 오프셋에 스택에 존재하는 값을 저장한다. 이 명령어를 수행하면 지역 변수의 첫 번째 오프셋에 현재 스택에 저장되어 있는 0 값이 저장된다. (그림 9)는 이러한 과정 및 결과를 시각적으로 보여주고 있다.



(그림 9) Istore_1 명령어 수행시 지역 변수 및 스택의 동작

reset 버튼을 선택한 경우 (그림 10)과 같이 PC가 어느 위치의 어떠한 명령어를 수행하든 상관없이 처음 PC 위치인 오프셋 0번으로 이동한다. 또한 스택과 지역 변수의 상태는 초기화 상태로 설정된다.



(그림 10) reset 버튼 선택

## 4. 결론 및 향후 연구 내용

본 논문은 자바 클래스 파일이 가지고 있는 정보를 이용하여 클래스 파일이 어떻게 구성되어 있는지를 단순한 텍스

트 형태가 아닌 시각화 형태로 보여주어 사용자가 필요한 정보에 대한 접근을 용이하게 함은 물론이고, 자바 가상 기계가 클래스 파일을 읽어들여 어떠한 명령들을 수행하여 스택과 지역 변수들이 어떻게 동작되는지에 대해 직접 실행시키며 살펴볼 수 있게 작성되었다. 이러한 연구는 차후 클래스 파일을 이용하여 다른 분야에 도움을 줄 수도 있으며 자바 가상 기계를 실행 분석기를 통해 실행해 봄으로 인해 좀 더 나은 프로그램의 최적화를 이룰 수 있을 것이다.

또한 본 논문 결과는 현재 다양한 분야에서 활용되고 있는 Java 언어를 위한 보다 효과적인 실행 환경 구축 및 개발 환경 개선에 활용될 수 있다.

## 참 고 문 헌


[1] Jon Meyer & Troy Downing, Java Virtual Machine, March, 1997.

[2] Tim Lindholm and Frank Yellin, The Java Virtual Machine Specification 2nd edition, Addision-Wesley, 1999.

[3] Bill Blunden, Virtual Machine Design and Implementation in C/C++, Wordware Publishing, Inc., 2002.

[4] Sun Microsystems, The K Virtual Machine(KVM) White Paper, Technical report, Sun Microsystems, 1999.

[5] Rainer Leupers and Peter Marwedel, Retargetable Compiler Technology for Embedded System : Tools and Applications, Kluwer Academic Publishers, 2001.

[6] John R. Levine, Linkers and Loaders, Morgan Kaufmann Publishers, 2000.

[7] Nik Shaylor, Douglas N. Simon, William R. Bush, A Java Virtual Machine Architecture for Very Small Devices, In the Proceddings of ACM SIGPLAN Conferences on Languages, Compilers, and Tools Embedded Systems 2003(LCTES '03), ACM Press, pp.34-41, 2003.

[8] John Whaley, Joeq : A Virtual Machine and Compiler Infrastructure, In the Proceddings of ACM SIGPLAN Conferences on Interpreters, Virtual Machines and Emulators 2003 (IVME '03), ACM Press, pp.58-66, 2003.

[9] Derek Rayside, Evan Mamas, Erik Hons, Compact Java Binaries for Embedded System, Proceddings of the 9th NRC/IBM centre for Advanced Studies Confercence (CASCON '99), pp.1-14, 1999.

[10] W. Paugh, Compressing Java class files, In the Proceedings of ACM/SIGPLAN Conference on Programming Language Design and Implementation(PLDI) '99, pp.247-258, May, 1999.

[11] Clausen, L. R., Schultz, U. P., Consel, C., Muller, G., Java Bytecode Compression for Low-End Embedded Systems, ACM TOPLAS, Vol.22, No.3, pp.471-489, May, 2000.

[12] Joshua Engel, Java Virtual Machine Programming, Infobook, 2000.

[13] 김도우, 정민수, 자바 프로그램 분석을 위한 바이트 코드 시뮬레이터, 정보처리논문지, 제7권 제7호, 2000.

[14] Jike Research Virtual Machine, http://www.ibm.com/developerworks/oss/jikesrvm.

[15] Waba Programming Platform, http://www.wabasoft.com.

[16] Joeq Virtual Machine, http://sourceforge.net/projects/jeoq.

[17] GTK Tools : http://www.gtk.org/tutorial.

[18] Artima Software, http://www.artima.com/insidejvm/applets/EternalMath.html.

[19] DJ Java Decompiler, http://neshkov.hit.bg/dj.html.

[20] Jad-the fast Java Decompiler, http://kpdus.tripod.com/jad.html.

[21] Class Cracker visual java decompiler, http://www.pcug.or.au.



**고 광 만**

e-mail : kkman@mail.sangji.ac.kr

1991년 원광대학교 컴퓨터공학과(공학사)

1993년 동국대학교 컴퓨터공학과 석사학위 취득(공학석사)

1998년 동국대학교 컴퓨터공학과 박사학위 취득(공학박사)

1998년~2001년 광주여자대학교 컴퓨터학부 전임강사

2002년~2003년 Queensland University of Technology 연구교수

2001년~현재 상지대학교 컴퓨터정보공학부 조교수

관심분야 : 프로그래밍 언어 설계 및 구현