《主　題》

인터네트에서의 차세대 표준언어 JAVA 개발환경

최 선 완
(한국전자통신연구소)

□차 례□

Ⅰ. 개요

OSI에 의해 사라질뻔 했던 인터네트는 웹(WWW)을 검색할 수 있는 브라우저인 Mosaic의 등장으로 회생은 물론 전세계적으로 폭발적 증가를 보이고 있다. 이제 많은 브라우저들이 등장하여 그 영역을 다투고 있고 모든 기능들이 웹으로 통합되는 플러그-인(Plug-In)화가 급속히 진행되고 있다. 그 결과 인터네트에 접속된 모든 사용자들은 브라우저만을 가지고 가상공간속에서 모든 문제를 해결하게 되었다. 한편, Java라 불리는 새로운 프로그래밍 개발 환경의 등장은 워크스테이션 시장의 한계로 인하여 그 발전 가능성이 불투명했던 선마이크로시스템즈(사)의 운명을 역시 반전시켰다. 본고에서는 Java의 주요개념과 특징에 대해서 기술한다.

Ⅱ. Java의 주요 개념

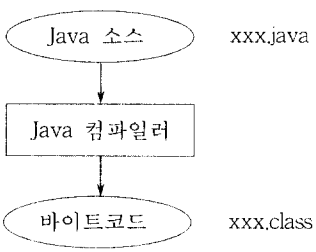
Java 프로그래밍 개발 환경(programming language environment)을 줄여서 Java라 한다. Java는 C 또는 C++와 같은 일종의 프로그래밍 언어이며 에니메이션, 네트워크, 게임등 모든 응용 프로그램을 개발하기 위한 일종의 도구에 불과하다. 그럼에도 불구하고 마이크로소프트(사)의 빌 게이츠까지 Java를 수용할 수 밖에 없었던 가장 중요한 원인은 Java로 작성된 모든 프로그램은 어떤 환경에 종속되지 않고 실행될 수 있는 바이트코드(byte codes)"라 불리는 가상머신(virtual machine) 개념을 도입했다는 것이다. 물론 Java 언어 그 자체에 대한 장점도 있으나 차후에 설명하도록 한다.

2.1 바이트코드(Byte Codes)와 Java 흐름도

어떤 프로그래밍 언어로 작성된 프로그램은 일단 컴파일 과정을 거쳐서 목적 코드(object code)를 생성하게 된다. 일반적으로 이때 생성된 목적 코드는 특정 기종에 종속된다. 웹 브라우저인 네트스케이프의 경우만 보아도 Sun의 Solaris 2.x.x, Sun OS 4.x.x 와 같이 동일한 Sun 플랫폼에도 불구하고 운영체제의 차이에 따라 새롭게 개발하여야 한다. 그러나, Java 개발 환경에서는 컴파일 과정후에 생성된 목적 코드가 모든 환경에 적용될 수 있도록 코드를 생성하는데 이를 바이트코드라 한다. 따라서, Java 환경은 Java 언어로 부터 바이트코드를 매핑하는 Java 가상머신(Virtual Machine)과 인스트럭션 집합을 정의하고 있다.

이러한 바이트코드 개념은 결국 이동성 코드(mobile code)의 개념이며, Java로 작성된 프로그램은 누가 어디서 어떤 환경에서 개발하거나 관계없이 수행될 수 있다는 확기적인 개념이다. 즉, Java를 이용하면 각 기종(PC, 워크스테이션, 매킨토시등)과 운영

체제(Windows95, SunOS, Solaris, MacOS등)에 대한 별도의 소프트웨어를 제공하지 않아도 된다. 따라서, 새로운 프로그램을 개발할때 이식성(portability)에 따른 개발노력의 단축뿐만 아니라 소프트웨어의 확산이 가하 폭발적으로 증가할 것이다. 그러나 Java가 바이트코드의 시초는 아니다. 1970년대 중반 부터 1980년대 초반까지 바이트코드 형태를 채택했던 UCSD P-System은 8비트 컴퓨터에 널리 사용되었으나, 그 처리능력의 한계로 제한될 수 밖에 없었다. 현재는 바이트코드 형태로 소프트웨어를 처리하고 배포하는데 충분한 컴퓨팅(네트워크 포함) 환경을 제공하고 있다. (그림 2-1)은 바이트코드 생성과정을 보여준다.

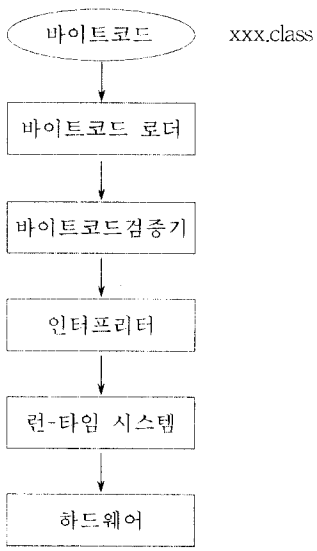


(그림 2-1) 바이트코드 생성과정

Java로 작성된 원천코드의 화일명은 확장자 .java가 항상 붙는다. Java 컴파일러에 의해 컴파일된 바이트코드의 화일명은 확장자 .class가 붙는다. 즉, 바이트코드는 객체지향 언어에서 부르는 한개의 클래스임을 나타낸다. C 개발 환경에서 C 컴파일러가 C 언어로 개발 되었듯이 Java 컴파일러 또한 Java를 이용하여 개발되었다.

2.2 애플릿(applet)

애플릿은 이동성 코드의 결정판이다. 애플릿이라 함은 HTML 내에서 작성된 Java 프로그램 또는 Java를 처리할 수 있는 웹 브라우저내에서 수행되도록 작성된 Java 프로그램이라고 정의한다. 즉, 브라우저에서 웹을 검색할때 HTML 내에서 〈applet〉 이라는 tag를 만나면 이때부터가 Java 프로그램임을 의미하며 〈/applet〉은 그 끝을 의미한다. 이때 애플릿은 서버에서 이미 컴파일되어 바이트코드로 생성되어 있는 상태로서 그 바이트코드는 웹 서버로 부터 브라우저로 다운로딩되고 브라우저는 그 바이트코드를 해석한다. 따라서, 브라우저 사용자는 자신이 특정 프로그램에 대한 코드를 갖지 않고도 즉시 원하는 프로그램을 수행할 수 있는 획기적인 개념인 것이다. 이것은 결국 바이트코드가 제공하는 코드의 중립성(neutral) 때문에 가능하다. (그림 2-2)는 애플릿이 브라우저(클라이언트)에서 수행되는 과정을 보여주며 런-타임에 발생함을 알 수 있다.



(그림 2-2) 애플릿 수행 과정과 Java 환경

바이트코드 로더는 웹 브라우저와 서버가 같은 시스템에 있으면 로칼 화일시스템으로 부터 바이트코드를 로딩하고 다른 시스템에 있으면 네트워크를 통해서 다운로딩한다. 바이트코드 검증기(verifier)는 로딩된 바이트코드의 구문이 맞는가를 검사한다. 바이트코드는 여러 Java 컴파일러에 따라 다른 결과를 생성할 수 있으므로 잘못된 컴파일 과정을 검사하는 것이다. 검증이 끝날때 까지 인터프리터에게 바이트코드는 전달되지 않는다. 그 인터프리터는 바이트코드를 해석하면서 즉시 수행한다. 이때 하드웨어 종속된 기능이 요구되며 런-타임 시스템이 그 작업을 수행한다. 런-타임 시스템은 POSIX-C와 일치하는 ANSI C 언어로 작성된다.

2.3 애플릿과 네트워크 컴퓨터

브라우저를 통한 웹의 검색은 정적인 클라이언트/서버 모델이다. 서버에서 제공하는 URL을 통해서 자료 검색등을 수행하고 많은 기능들이 클라이언트(브라우저)에서 외부 뷰어를 갖추고 있어야 한다. 그러나 애플릿은 그 코드가 서버에서 다운로딩되므로 클

라이언트(브라우저)는 단지 인터프리터만 있으면 모든 Java 프로그램을 수행시킬수 있다. 이 개념은 최근에 관심 대상이 되고 있는 네트워크 컴퓨터의 개념과 같다. 즉, 네트워크 컴퓨터는 애플릿 뿐만 아니라 운영체제, 통신등 모든 코드를 서버로 부터 항상 다운로딩한 코드를 이용하기 때문에 클라이언트에 프로그램이 없어도 되는 무구현(no implementation) 개념이다.

Ⅲ. Java 개요

2장에서는 Java의 중요한 특징인 바이트코드와 애플릿에 대한 개념을 살펴 보았다. 본 장에서는 Java의 특성을 기술한다.

3.1 Java의 역사: 숨겨진 이야기

Java를 이해하기 위해서 먼저 그 탄생 배경을 살펴본다. 아무리 뛰어난 연구도 그 시대의 정치, 경제, 기술적인 상황과 일치할때 비로소 빛을 보게된다. Java의 최근 관심도 결국 웹의 폭발적인 확산이 없었다면 사장될 뻔한 시대를 앞서는 개념이었다. 특히, Java의 숨겨진 이야기는 우리의 개발 환경 현실을 다시금 돌아보게 한다. Java는 1990년 선 마이크로시스템즈(사)의 James Gosling에 의해서 시작되었고 가정용 전자제품(1991년)과 쌍방향 TV(1993년)에 이를 적용하였다. 그러나 그 기술의 우월성에도 불구하고 실패할 뻔한 프로젝트였으나 네트스케이프에서 Java를 수용함으로써 빛을 보고 있는 셈이다.

James Gosling은 IBM에서 1984년 Sun으로 옮겼고 NeWS 윈도우 시스템과 EMACS 에디터 프로젝트에 참여하였다. 1990년 초에 Sun은 고가이면서 많은 power를 필요로 하는 워크스테이션 시장보다는 가전제품 시장에 참여하기를 바랬다. 따라서 가정용 가전제품을 보다 신뢰성있고, 저렴하고, 표준화되고, 단순하고, 실시간이고, 개발하기 쉬운 환경을 위한 Green 프로젝트에 James Gosling이 참여하게 되었다. 가전제품은 흑백과 컬러 TV의 backward compatibility가 보장되고 긴 수명을 갖는 특성을 갖는다. 1991년 중반, Green 프로젝트에서는 어떠한 플랫폼에도 종속되지 않는 개발 환경을 위해서 C++를 확장하였으나 충분치 않아 가전제품의 대규모, 분산, 이종의 네트워크 환경에 적합한 Oak를 개발 하였다. 1992년 가을에 Sun의 Open Windows 프로젝트 책임자이던 Patrick Naughton을 책임자로 Oak, Green OS, 사용자 인터페이스, 하드웨어에 대한 관점에서 리모트 콘트롤 한개로 모든 가전 제품을 조작할수 있는 *7이라는 하드웨어 디바이스를 만들기 위해서 약 75명의 연구원이 투입되었다. 1993년 초에 타임 워너(사)로 부터 셋톱 박스 운영체제와 VOD 기술에 대한 제안을 받고 Sun은 FirstPerson Inc.를 설립하였으나 SGI의 로비로 프로젝트를 받지 못해 1994년초에 대부분이 Sun Interactive의 디지탈 비디오 데이터 서버 개발팀에 합류하였고 나머지는 쌍방향 TV에 대한 연구를 계속하였으나 시장이 성숙되지 않았다. 한편 그 일부는 계속 Oak를 이용한 연구를 진행하였는데 1993년 등장한 Mosaic과 1994년 네트스케이프의 폭발적 증가로 방향을 선회하여 웹을 목표로 Liveoak 프로젝트를 시작해서 Oak를 이용한 웹 브라우저인 WebRunner(후에 HotJava로 불림)를 1994년 가을 첫번째 데모를 가졌다. 이 과정에서 Oak라는 이름은 버리고 James Gosling과 Arthur van Hoff는 Java와 Java compiler를 개발하였고 Sun은 1995년 5월 SunWorld 95에서 공식으로 Java와 HotJava를 발표하였다. Java는 최초 알파 버전으로 출발하여 1996년 1월 현재 정식으로 자바개발환경(JDK1.0, Java Developers Toolkit)을 발표하였다. HotJava는 Java로 작성된 웹 브라우저로서 알파 버전으로 개발되었으며 현재 JDK1.0으로 개발중이다.

3.2 Java의 특성

Java의 기본 모델은 C++이며 기존의 Effel, SmallTalk, Objective C, Cedar/Mesa로 부터 많은 아이디어를 얻었으며 그 결과 네트워크 장치로 부터 웹, 데스크톱에 이르는 다양한 환경에서 적용할 수 있으며 그 특징은 1) 단순하고(simple) 친숙하며(familiar), 2) 객체지향(object-oriented) 언어이고, 3) 시스템에 독립적이고(architecture neutral, portable, robust), 4) 인터프리트 형이고(interpreted) 동적이며(dynamic), 5) 안전성을 보장하고(secure), 6) 멀티스레드를 지원하고(multithread), 7) 다른 언어와 비교하여 고성능이다(high performance).

3.2.1 단순성(simple)과 친숙성(familiar)

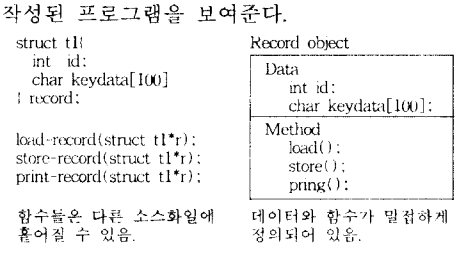
Java에서는 기존의 C를 배우는 노력과 C++와 같은 객체지향언어에 서의 사용되지 않거나 혼동스럽거나, 이해하기 어려운 부분은 생략하였다. 또한 기존의 C와 C++와 유사하기 때문에 쉬게 이용할 수 있다. Java 언어는 일반 언어와 마찬가지로 주석(//, /* */,

//* */), 프리미티브 데이터 타입(integer, floting point, character, boolean, string), 수치와 관계형 연산자(++, --, !, ~, *, /, %, +, -, <<, >>, >>>, <>, <=, >=, ==, !=, &, |, &&, ||, =, *=, +=, -=, /=, %=), 배열([]), multi-level break(if else, switch, for, while, do while, break, continue, try-catch)를 제공한다. 특히, C와 C++에서는 메모리를 할당 받거나(memory allocation) 해제할때(release) 직접 구현자가 프로그램에 명확히 작성해야 하지만 Java에서는 자동적인 메모리 관리를 수행한다. 즉, C에서의 malloc과 free와 같은 함수를 이용할 필요가 없다. 이러한 메모리의 처리(garbage collection)는 사용자가 쉬고 있는(idle) 상태에서 수행하므로 그 효과가 크다. C와 C++에서 제거된 Java의 기능들은 다음과 같다: 1) #typedef, #define이 없다. 2) union과 struct 타입이 없다. 3) 객체지향 기법을 이용하여 function과 procedure 기능을 제거할 수 있다. 4) 다중 상속(multiple inheritance)를 허용하지 않는다. 5) 타입 캐스트를 허용하지 않는다. 6) 포인터(pointer)가 없다.

3.2.2 객체지향(object-oriented)

객체지향에 대한 개념은 다양한 분야에서 사용되고 있으므로 자세한 내용은 생략한다. 클래스는 데이터와 그 데이터상의 동작(operation)을 포함한 데이터 타입을 말한다. 여기서 동작은 함수(function)를 가리킨다. 객체란 클래스가 구체화된 경우, 즉 인스턴스를 말한다. 메쏘드(method)는 클래스의 함수(function)/동작(operation)/행위(behavior)을 가리킨다.

클래스는 상위 클래스(superclass)로 부터 상태(state)와 행위를 상속받으며 이것은 프로그래밍을 구조화는데 매우 강력한 수단을 제공한다. 따라서, 모든 클래스는 어떤 클래스로부터 그 특성을 부여받고 이를 이용하는데 이를 클래스 계층구조로서 나타내게 된다. (그림3-1)은 C로 작성된 프로그램과 객체형태로 작성된 프로그램을 보여준다.



(그림 3-1) 객체의 한 예

객체지향 설계는 인터페이스의 명확한 정의와 소프트웨어의 재사용을 제공한다. 특히 분산 클라이언트-서버 모델에 아주 적합한 모델이다. 즉, 어디선가 생성된 객체는 네트워크를 통해서 전달되고 어디선가 저장되어 있다가 작업시에 추출하여 이용하게 된다. Java에서의 객체지향 기능들은 Eiffel, SmallTalk, Objective C, C++의 장점을 이용하였다. 특히, C++로 부터 객체 모델을 확장하였고 앞장에서 기술한 많은 기능들은 제거하였다. Java에서 제공하는 객체지향은 최소한 다음 4가지 특성을 지원한다.

- 캡슐화(encapsulation): 정보 은닉(hiding)과 모듈화 제공
- polymorphism: 동일 메시지라도 다른 객체에 전송되면 그 행위는 받은 객체의 특성에 종속된다.
- 상속성(inheritance): 코드를 재사용하고 구축하기 위해서 기존 클래스에 근거하여 새로운 클래스와 행위(behavior)를 정의할 수 있다.
- 동적 바인딩(dynamic binding): 객체는 어디서나 올수 있으며 특히 네트워크를 통해서 올수도 있다. 따라서 프로그램이 수행되는 도중에 바인딩되며 최대한의 융통성을 제공한다.

3.2.3 중립성(architecture neutral), 이식성(portable), 견고성(robust)

네트워크의 폭발적인 증가는 컴퓨터에서 수행되는 응용 또는 응용 일부가 다양한 컴퓨터 시스템, 다양한 하드웨어 구조, 다양한 운영체제, 다양한 윈도우 시스템에 쉽게 이식되기를 요구한다. 그러나, 기존의 방법은 특정 목적 시스템에 종속된 소프트웨어를 개발하고 그 이진코드(binary code)들을 배포하였으므로 자신의 기종에 맞는 소프트웨어가 나올때까지 기다려야만 하였다. 이러한 문제를 해결하기 위한 방법이 특정 시스템에 종속되지 않는 이진코드 형식이며 이를 기종 중립(architecture neutral)이라 한다. 이를 해결한 것이 바이트코드이다. Java 컴파일러는 특정 머신코드를 생성하지 않고 상위 단계의 일종의 가상의 머신 독립적인 코드인 바이트코드를 생성한다.

인터프리트된 바이트코드 방법의 큰 장점은 컴파일된 Java 프로그램이 Java 인터프리터와 런-타임 시스템이 구현된 어떤 시스템에도 이식(portable)될 수 있다는 것이다. 기존의 C와 C++로 프로그래밍을 할때 프로그래머는 시스템에 종속적인 사항에 대하여 별도의 구현을 요구한다. Java에서는 모든 플랫폼마다 데

이터 타잎을 새롭게 적용하는 것을 제거하기 위하여 다음과 같이 모든 프리미티브 데이터 타잎과 그 행위를 정의하고 있으며 모든 Java 구현에 표준이다.

byte	8-bit 이진보수
short	16-bit 이진보수
int	32-bit 이진보수
long	64-bit 이진보수
float	32-bit IEEE 754 부동점
double	64-bit IEEE 754 부동점
char	16-bit Unicode 문자

Java는 견고한(robust) 언어이다. Java는 C와 같은 묵시적(implicit)인 타잎 선언을 허용하지 않는 강력한 타잎 언어이다. 따라서, 컴파일 시간에 대부분의 타잎 에러를 발견하여 실행시간(run-time)에 발생하는 에러를 최소화한다. C/C++와 Java의 큰 차이는 Java에서는 포인터 방식 대신에 배열(array)만을 갖기 때문에 포인터에 의해 메모리가 중복되고 데이터가 깨지는 가능성을 제거한다.

3.2.4 인터프리트되고(interpreted) 동적임(dynamic)

기존의 소프트웨어 개발주기(life-cycle)는 에디트-컴파일-링크-로드-런 과정을 거치며 컴파일 시간에 모든 에러를 찾지 못했다. Java환경에서는 모든 하드웨어와 운영체제에서 Java인터프리터를 요구하지만 한번 그 Java 인터프리터와 런-타임 시스템을 갖추면 모든 Java프로그램을 실행시킬 수 있다. Java에서 링크는 새로운 클래스를 로딩하는 과정을 의미한다.

Java의 이식성과 인터프리트되는 특성은 매우 동적인 시스템을 제공한다. 클래스는 요구하는 싯점에 네트워크로 부터 다운로딩되어 링크될 수도 있다. Java의 또 한가지 특징은 C++와 달리 수퍼-클래스 문제가 발생하지 않는다. C++에서는 클래스에 새로운 메쏘드(method)와 인스턴스(instance)를 추가하면 그 클래스를 참조한 모든 클래스는 다시 재컴파일을 요구하지만 Java는 클래스 바인딩을 런-타임에 수행한다. 또한 Java는 컴파일시에 객체의 저장소 배치(storage layout)를 결정하지 않고 런-타임시에 인터프리터가 결정한다.

3.2.5 안전성(security)

Java에서의 안전성은 일반 암호화와는 구별된다. Java에서는 Java 컴파일러와 런-타임 시스템이 잘못된 코드를 생성하는 응용 프로그래머를 어떻게 제한할 것인가의 관점에서 안전성을 추구한다. 첫번째 방법은 앞에서 기술한 바와 같이 메모리 할당을 런-타임에 실행하고 포인터를 사용하지 않음으로써 컴파일 타임에 생성된 바이트코드가 메모리를 파괴하는 것을 막을 수 있다. 두번째는 바이트코드 검증기를 이용하여 잘못된 바이트코드가 수행되기 전에 발견하는 방법이다. 세번째는 바이트코드 로더가 바이트코드를 로칼 화일시스템과 각 네트워크로 부터 로딩할때 발생하게 될 클래스(바이트코드)의 중복을 방지할 수 있는 이름영역(name space)을 두는 방법이다. 네번째는 네트워킹에 따른 문제이다. 즉, 브라우저와 서버가 다른 시스템에 있는 경우에 애플릿(applet)은 서버로부터 네트워크를 통해서 브라우저로 다운로딩되는데 이때 애플릿이 클라이언트 시스템을 파괴할 수 없도록 다음과 같은 제약을 두고 있다.

- 애플릿은 클라이언트의 화일 시스템을 접근할 수 없다.
- 애플릿은 클라이언트에서 프로세스를 생성할 수 없다.
- 애플릿은 서버를 제외한 다른 컴퓨터에 접근할 수 없다.

결국 Java에서 안전성을 보장한다는 의미는 언어 자체가 완전한 타잎 정의를 요구하여 컴파일 과정에 명확한 바이트코드를 생성함으로써 잘못된 코드가 생성되는 것을 크게 줄일수 있다는 것이다. 또한 바이트코드 검증기가 다시 바이트코드를 검사하기 때문에 완전한 안전성을 보장한다.

3.2.6 멀티스레드(multithread) 지원

스레드란 한 프로세스내에서 독립적인 제어가 가능한 흐름이라고 정의한다. 보통 execution context 또는 lightweight 프로세스라 한다. 이런 분구상의 정의에도 불구하고 한개의 스레드로 구성된 프로세스에서 스레드와 프로세스의 차이는 없다. 단지 멀티스레드에서는 스레드들간에 메모리를 공유한다는 차이는 있다. 멀티스레드는 프로세스간 통신이보다 쉽고 특정 코드를 공유함으로써 부하를 줄이는 lightweight 프로세스를 지원한다.

Java 또한 이러한 멀티스레드 기능을 지원함으로써 서버로 부터 데이터를 다운로딩하면서 페이지를 넘기는 도중에 음악을 듣고 에니메이션을 수행하는 대화형 그래픽 응용 프로그램을 효과적으로 개발할 수 있

다. Java 스레드는 java.lang 패키지(3.2장 참조)의 일부인 Thread 클래스를 구현한 것이다. 이 Thread 클래스는 시스템 독립적인 Java 스레드를 구현한 것이며 실제로의 동작은 시스템에 종속된다. Java에서 새로운 스레드를 생성하기 위해서 new Thread( )를 호출한다. Java는 멀티스레드를 위한 스레드의 관리, 우선순위 책정(MIN_PRIORITY와 MAX_PRIORITY 사이), 동기화를 위한 클래스와 메쏘드를 제공하며 중요한 메쏘드들은 다음과 같다.

start( ) 메쏘드: 스레드를 실행사킬때 필요한 시스템 자원을 생성하고 스레드가 실행될 준비를 한다.

stop( ) 메쏘드: 스레드를 정지한다.

run( ) 메쏘드: 스레드에서 가장 중요한 부분이다. 스레드가 생성되고 시작된 후에 스레드의 행동을 결정하는 스레드의 몸체(body) 부분이다. 즉, 스레드에서 무한 루프를 돌면서 입출력을 행하거나, 스레드를 잠시 정지시킨 후에 다시 재개시키는등의 기능들을 run( ) 메쏘드안에서 지정할 수 있다.

suspend( ) 메쏘드: 잠시 스레드를 정지한다.

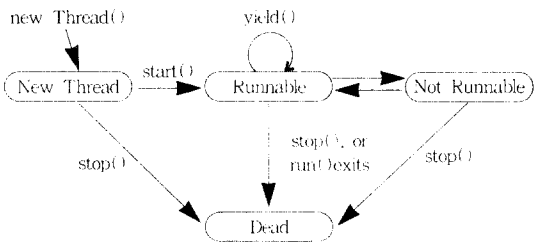
resume( ) 메쏘드: 정지된 스레드를 재개한다.

sleep( ) 메쏘드: 1000분의 1초 단위로서 스레드를 정지한다.

wait( ) 메쏘드: 조건 변수(condition variable)의 상태가 true가 되면 스레드를 깨운다.

yield( ) 메쏘드: 현재 실행중인 스레드를 멈추고 다음 스케쥴링시에 실행시킨다.

(그림 3-2)는 Java 스레드의 상태천이도이다. New Thread 상태는 시스템 자원을 할당받지 못한 빈 Thread 객체를 의미한다. 그 Thread 객체는 start( ) 메쏘드에 의해 시스템 자원을 할당받고 실행되도록 스케쥴링된다. 이때 start( ) 메쏘드는 run( ) 메쏘드를 호출한다. 특기할 사항은 스레드가 Running이 아닌 Runnable 상태로 간다는 것이다. Runnable의 의미는 스레드가 생성되고 시작되어도 실제로 한개의 CPU를 갖는 환경에서는 즉시 그 CPU를 할당받지 못하여 기다리고 있는 상태를 의미한다. 스레드는 suspend( ) 메쏘드, sleep( ) 메쏘드, wait( ) 메쏘드, I/O 블럭킹중의 한가지 경우에 Not Runnable 상태로 간다. Runnable 상태로의 복귀는 resume( ) 메쏘드 또는 조건변수들의 상태를 알리는 notify( )/notyfyAll( ) 메쏘드, 또는 I/O가 해제된 경우에 발생한다. 만일 스레드의 현재 상태를 알기 위해서는 isAlive( ) 메쏘드를 이용하며 New Thread와 Dead인 경우에는 false 값이 Runnable과 Not Runnable인 경우에는 true 값이 리턴된다.



(그림 3-2) Java 스레드의 상태천이도

3.2.7 고성능(high performance)

Java 개발환경의 큰 특징은 코드가 매우 작다는 것이다. 인터프리터와 클래스의 크기가 약 40K 바이트면 충분하고 마이크로 커널에서 이미 지원하는 기타 라이브러리와 스레드를 위해서 추가적으로 175K 바이트만이 요구된다. 특히, 바이트코드 양식이 매우 단순하여 처리하기가 쉽다. (표 3-1)는 기능적인 측면에서 Java와 다른 언어를 비교한 예이다.

	Java	SmallTalk	TCL	Perl	Shells	C	C++
Simple	●	●	●	◑	◑	◑	○
Object Oriented	●	●	○	●	○	○	◑
Robust	●	●	●	●	●	○	○
Secure	●	◑	◑	●	◑	○	○
Interpreted	●	●	●	●	●	○	○
Dynamic	●	●	●	●	◑	○	○
Portable	●	◑	●	●	◑	◑	◑
Neutral	●	◑	◑	●	◑	○	○
Threads	●	○	○	●	○	○	○
Garbage Collection	●	●	○	○	○	○	○
Exceptions	●	●	○	●	○	○	◑
Performance	High	Medium	Low	Medium	Low	High	High

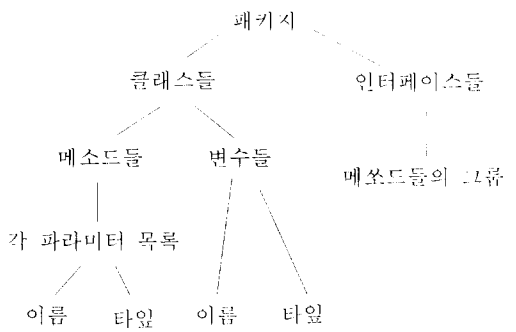
● Feature exists ◑ Feature somewhat exists ○ Feature doesn't exist

(표 3-1) Java와 다른 언어의 기능 비교[2]

3.3 Java API

Java로 작성된 모든 프로그램은 특정 클래스를 생

성한다. Java는 프로그래머에게 다양한 클래스를 제공하는데 이 클래스들과 인터페이스들이 모여서 특정 패키지가 된다. Java API는 패키지(pacakge)에 대한 프로그래밍 인터페이스를 말한다. (그림 3-3)은 한 패키지를 구성하는 계층도를 보여준다.



(그림 3-3) Java 패키지 계층도

객체지향 언어인 Java는 클래스들의 특성에 따라 다음 6 종류의 API 패키지(바인더 3권 분량)를 제공한다. 그러나 이들 패키지는 더 세분화되어 8 종류로 나누기도 한다.

java.lang 패키지: Java 언어에서 built-in으로 제공되는 클래스이다. Boolean, character와 같은 일반 프로그래밍 언어의 built-in 기능과 스레드 관련 클래스, 그리고 예외상황(exception handling)에 관련된 클래스들로 구성된다.

java.awt 패키지: AWT는 Abstract Window Toolkit의 약자로 윈도우를 구성하기 위한 체크박스, 버튼, 패널, 스크롤, 레이아웃등의 클래스와 이미지 클래스들(java.awt.image 패키지)과 애플릿 클래스들(java.applet 패키지), 그리고 윈도우를 위한 인터페이스들(java.awt.peer 패키지)을 포함한다.

java.io 패키지: 입출력과 관련된 클래스와 인터페이스로 구성된다.

java.net 패키지: 네트워크 프로그래밍을 위한 소켓(socket)등의 클래스와 URL 처리를 위한 클래스로 구성되고 관련 인터페이스들도 포함된다.

java.util 패키지: Java 프로그래밍을 하는데 유용한 데이트, 사전, 스택등과 같은 클래스와 인터페이스들로 구성된다.

sun.tool.debug 패키지: Sun에서 제공하는 디버깅을 위한 클래스와 인터페이스들을 포함한다.

Ⅳ. Java 개발환경(JDK: Java Developers Kit)

Sun은 Java 프로그래밍을 위하여 JDK1.0을 1996년 1월에 발표하였다. JDK에서 제공하는 도구를 사용하기 위해서는 먼저 환경을 지정해야 한다. 첫째로 JDK에서 제공하는 API 클래스들이 있는 디렉토리(path)를 setenv CLASSPATH pathname:pathname:..로 지정해야 한다.

Java 인터프리터는 바이트코드를 처리하면서 필요한 클래스들을 지정된 path로 부터 찾는다. 또한 독자적으로 개발된 클래스에 대해서도 위치한 데릭토리의 path를 지정하면 인터프리터는 지정된 순서대로 찾는다. JDK1.0에서 Java 프로그래밍을 위한 도구들은 다음과 같다.

javac: Java 컴파일러이다. Java로 작성된 프로그램(*.java)을 컴파일하여 바이트코드(*.class)를 생성한다.

javac_g: 디버깅이 가능하도록 Java 프로그램을 컴파일하여 바이트코드를 생성한다.

java: Java 인터프리터이다. 바이트코드를 해석하고 수행한다.

java_g: 디버깅을 위한 Java 인터프리터이다.

javah: 기존의 다른 프로그램과의 링크를 지원하기 위해서 바이트코드(클래스 화일)로부터 헤더화일(*.h)과 소스화일(*.c)을 생성한다. Java에서는 이것을 native 메쏘드라 한다.

javah_g: 디버깅을 위한 헤더와 소스화일을 생성한다.

javaprof: java 명령어를 수행시 -prof 옵션을 주면 *.prof 화일이 생성된다. 이 화일은 메쏘드와 클래스의 호출 횟수, 데이터 타잎에 대한 메모리 사용등 통계자료를 위한 정보를 포함한다.

javap: Java 프로그램으로 부터 클래스의 각 필드를 옵션에 따라 분리한다.

javadoc: Java 프로그램으로 부터 하이퍼텍스트 형태로 API 문서를 생성한다. 일반적인 API 문서형태는 클래스와 인터페이스 부분, 메쏘드, 그리고 변수(variables)로 구성된다.

appletviewer: 웹 브라우저를 이용하지 않고 명령어로 애플릿을 실행시킨다.

jdb: Java 프로그램에 대한 디버깅 도구이다.

4.1 Java 프로그래밍

Java 프로그램은 Java 응용(application)과 Java 애

플릿으로 구분된다. Java 응용은 독자적인 Java 프로그램으로 main( )이 있다. 애플릿은 HTML내에서 작성된 Java 프로그램으로 main( )이 없다. 간단한 Java 응용과 애플릿의 예로서 HelloWorld 클래스가 주로 사용된다.

4.1.1 Java 응용: HelloWorld

```
// HelloWorld.java
class HelloWorld
public static void main(String args[])
Systems.out.println(Hello World);}
```

(그림 4-1) Java 응용 예: HelloWorld

(그림 4-1)은 HelloWorld 클래스를 HelloWorld.java 화일에 구현하여 저장한 Java 응용의 예이다. 이때 클래스 이름과 화일 이름이 같을 필요는 없다. Java 인터프리터(java)는 단지 클래스 이름을 이용하여 수행한다. //는 주석을 나타낸다. HelloWorld 클래스는 main( ) 메쏘드를 포함한다. C언어와 같이 그 main( ) 메쏘드는 프로그램이 시작되는 지점이다. 이때 main( ) 메쏘드는 public 이라는 접근 수단을 제공하는데 누구나(어디서나) 호출할 수 있다는 의미이다. 만일 protected로 선언되면 subclass에 의해서 호출될 수 있고 private는 그 클래스만이 호출할 수 있다. 또한 C와 같이 void는 리턴값이 없음을 나타낸다. 명령어상의 인자(argument)는 main( ) 메쏘드에게 args라는 스트링 배열로 전달된다. 이 화일를 컴파일하면 HelloWorld.class 화일이 생성된다. 명령어 java HelloWorld에 의해 HelloWorld.class의 바이트코드가 해석되고 처리된다. C 언어인 경우에 라인상의 커맨드의 인자args[0]은 java이고 args[1]은 HelloWorld가 main에 전달되지만 Java에서는 java HelloWorld test인경우에args[0]은 test를 의미한다. System.out.println은 3.3장에서 기술한 Java API 중에서 java.lang 패키지내에 system 클래스의 한가지로서 C에서의 printf과 같은 의미를 같는다. 따라서 본 Java 응용의 결과는 Hello World가 화면에 나타난다.

4.1.2 HelloWorldApplet: Java 애플릿

(그림 4-2)는 HTML 내에서 HelloWorldApplet 클래스를 포함한 Java 애플릿의 예이다. Java 애플릿이 수행되기 위해서는 먼저Java 원천코드인 HelloWorldApplet.java 화일을 컴파일해야 한다. 다음에 그 애플릿을 포함한 HTML 화일을 만들어야 한다. 웹 브라우저가 (그림 4-2)의 HTML 화일에서 〈APPLET〉 태그을 만나면 애플릿의 시작임을 안다. codebase는 애플릿이 어디에 있는가를 나타내는데 디폴트 path는 setenv CLASSPATH에 정의된 디렉토리이다. 여기서는 디폴트 디레토리 아래에 있는 classes 디렉토리에 있다는 의미이다. code는 찾을 클래스 화일 이름을 가리킨다. HelloWorld.class 화일을 찾으면 그 바이트코드는 네트워크를 통해서 또는 로칼 화일시스템으로부터 브라우저에게 로딩된다. 이때 브라우저내에서 애플릿이 수행될 영역을 width와 height로서 지정해야 한다. 또한 파라미터를 애플릿에 전달할 수가 있는데 paramname에 이름을 명시하고 value에 그 값을 명시한다. 즉, 애플릿이 수행중에 Howdy there!가 애플릿에 전달된다. (그림 4-3)은 HelloWorldApplet 클래스의 예이다.

```
〈HTML〉
〈HEAD〉
〈TITLE〉 Hello World 〈/TITLE〉
〈/HEAD〉
〈BODY〉
This is the applet:
〈APPLET codebase=classes
code=HelloWorldApplet.class
width=200 height=200〉
〈paramname =message value=Howdy there!〉
〈/APPLET〉
〈/BODY〉
〈/HTML〉
```

(그림 4-2) 애플릿을 포함한 HTML

```
// HelloWorldApplet.java
import java.applet.Applet
import java.awt.Graphics
public class HelloWorldApplet extends Applet
String input__from__Page;
public void init( )
System.out.println(initializing..);
input__from__Page=getParameter(String);
public void start( )
System.out.println(starting...);
public void stop( )
System.out.println(stopping..);
```

```
publci void destroy( )
System.out.println(preparing for unloading...);
public void paint(Graphics g)
g.drawString(Hello world!, 50, 25);
```

(그림 4-3) Java 애플릿: HelloWorldApplet 클래스

(그림 4-3)의 두번째 줄에 import는 관련 패키지의 클래스들을 HelloWorldApplet 클래스에서 이용한다는 의미이다. 따라서 public … extends Applet은 HelloWorld 클래스가 Applet 클래스의 특성을 상속받겠다는 뜻이다. 다른 방법으로는 import java.applet.Applet을 정의하지 않는 대신 public class … extends java.applet.Applet으로 사용해도 무방하다. 다음은 스레드에서 설명한 메쏘드들이 이용되는데 우리가 브라우저에서 애플릿을 만난 경우를 보자. 다음 3단계를 거치게 될 것이다.

단계 1: 애플릿은 네트워크 또는 로칼 화일시스템으로 부터 로딩된다. 애플릿이 로드될때 먼저 그 애플릿이 제어하는 클래스의 인스턴스가 생성된다. 다음에 그 애플릿을 초기화(init( ) 메쏘드 이용)한다. 다음에 start( ) 메쏘드를 호출하여 애플릿을 실행모드로 이동시킨다.

단계 2: 애플릿이 자신의 브라우저에서 수행되는 도중에 우리는 다른 웹 페이지로 가거나 그 웹에서 빠져나올 것이다. 다른 페이지로 이동하는 경우에 그 애플릿의 수행을 멈출 필요가 있는데 stop( ) 메쏘드를 이용한다. 그 후에 다시 그 페이지로 돌아오면 start( ) 메쏘드를 호출하여 애플릿을 다시 실행시킨다. 사용자는 그 페이지를 재로드(브라우저에서 reload 버튼)를 할 수도 있다. 이때 stop( ) 메쏘드를 호출한후에 destroy( ) 메쏘드를 호출하여 그 애플릿에 할당되었던 모든 자원을 해제한 후에 1)의 과정을 다시 수행하여 애플릿을 새롭게 수행시킨다(unloaded & reloaded).

단계 3: 마지막으로 우리는 브라우저를 완전히 빠져나올 것이다. 이때 stop( ) 메쏘드와 destroy( ) 메쏘드를 이용하여 모든 자원을 해제하게 되며, 그 후에야 비로소 브라우저에서 Are you really exit ? 같은 행동을 취하게 된다.

(그림4-3)에서 input_from_Page-get Parameter (String): 묻은 (그림 4-2)의 HTML 문서내의 애플릿내에서 〈paramname = message value Howdy there!〉에 따라 전달된 스트링 Howdy there!를 읽어들이는 getParameter( ) 메쏘드의 예이다. (그림 4-3)에서 paint( ) 메쏘드는 java.awt.Graphics 클래스의 한 메쏘드로서 브라우저에서 애플릿을 표현하는 기본적인 디스플레이 메쏘드이다. (그림 4-3)에서 g.drawString(Hello world!, 50, 25); 은 스트링 Hello world!를 x 좌표 50, y 좌표 25에 그리라는 메쏘드이다.

4.2 Native 메쏘드

Java 환경에서 메쏘드를 다른 프로그램으로 구현할 수 있으며 이를 native 메쏘드라 한다.

그 순서는 다음과 같은 7단계를 거친다.

단계 1: Java 코드를 작성한다: 아래와 같이 HelloWorld라는 Java 클래스를 만들고 그 클래스에서 displayHelloWorld( )라는 native 메쏘드를 선언한다.

```
// HelloWorld.java
class HelloWorld
public native void displayHelloWorld( );
static
System.loadLibrary(hello);
```

또한 다음과 같이 HelloWorld 객체를 생성하는 메인 프로그램을 만들고 그 native 메쏘드를 호출한다.

```
// Main.java
class Main
public static void main(String args[ ] )
new HelloWorld( ).displayHelloWorld( );
```

단계 2: Java 프로그램을 컴파일한다: javac HelloWorld.java; javac Main.java

단계 3: HelloWorld Java 클래스로 부터 C 헤더화일을 생성한다: javah HelloWorld. 생성된 화일 이름은 HelloWorld.h 이다. 그 헤더화일은 HelloWorld 클래스를 나타내는 구조를 정의하고, 또한 그 클래스에서 정의한 native 메쏘드displayHelloWorld( )의 구현에 대한 C 함수정의(function definition)를 제공한다.

단계 4: 스터브(stub) 화일을 생성한다: javah -stubs HelloWorld. 스터브 화일은 Java 클래스와 그 클래스의 병렬 C 구조를 포함하며 화일 이름은 HelloWorld.c 이다.

단계5: Native 메쏘드 Hello World_display Hello World( )에 대한 C 프로그램을 작성한다. 본 예에서는 화일 이름을 HelloWorldImp.c 이다.

```
/* HelloWorldImp.c */
#include 〈StubPreamble.h〉
```

```
#include HelloWorld.h
#include <stdio.h>
void HelloWorld__displayHelloWorld(struct
Hhellowprld *this)
printf(Hello World!\n);
return;
```

단계6: 그 native 메쏘드가 라이브러리를 이용한다면 런-타임에 이 라이브러리를 로드하기 위해서 메쏘드 loadLibrary( )를 작성한다.

단계 7: C 코드와 라이브러리를 컴파일한다: cc -g HelloWorld.c HelloWorldImp.c -o libhello.so

단계 8: 실행시킨다: java Main. Hell World! 가 화면에 나타난다.

Ⅴ. Java 애플릿 제약점과 네트스케이프(Netscape)

애플릿의 장점은 바이트코드에 의한 코드 다운로딩 개념이다. 앞에서 기술한 바와 같이 서버로 부터 다운로딩 되는 코드로 부터 안전성을 보장하기 위하여 애플릿은 다음과 같은 제약사항과 특징을 갖는다.

- 애플릿이 서버에 프로세스를 생성하는 것은 가능하지만 상당히 어렵다.
- 애플릿은 클라이언트에서 프로세스를 생성할 수 없다.
- 애플릿간의 통신은 동일 페이지에서는 가능하지만 다른 페이지에서는 불가능하다.
- 애플릿으로 부터 오디오를 기록할 수 없다.
- 애플릿이 네트워크를 통해서 다른 컴퓨터와의 접속을 개방할 수 없다. 예외적으로 애플릿 태그안에 codebase에 명시된 HTML 페이지는 가능하다.
- 애플릿이 만든 윈도우와 Java 응용이 만든 애플릿과는 구별되도록 한다.

애플릿은 네트워크로 부터 다운로딩되는 경우와 로칼 화일시스템으로 부터 로딩되는 경우로 분류되는데 그 차이점은 다음과 같다.

1) 네트워크로 부터의 로딩:
- 애플릿 클래스 로더에 의해서 로드됨
- 애플릿 시큐리티 관리자에 의해 제한됨

2) 로칼 디스크로 부터의 로딩:
- 화일 시스템 로더에 의해서 로드됨
- 화일 읽기와 쓰기 허용
- 클라이언트상에 라이브러리의 로드 허용
- 프로세스 생성 허용
- 가상머신의 탈출(exit) 가능
- 바이트코드 검증기를 통하지 않음

한편 Java 애플릿은 이를 처리하는 브라우저에 의해서 모든 기능이 지원되지 않을 수 있

다. 현재 네트스케이프에서는 Java 애플릿에 대한 다음 문제점을 갖는다.

- 애플릿 로딩시에 네트워크로 부터의 로딩은 http:URL을 이용하고 로칼 화일시스템으로 부터의 로딩은 file:// 또는 ftp:URL을 이용한다.
- 소켓(socket)은 애플릿이 다운로드된 머신에 열게되며(open) 인터네트상에서는 안된다.
- 네트스케이프 Mac과 Windows3.1버젼에서는 Java를 지원하지 않는다.
- 웹 페이지를 재로드(reload)할때 캐쉬로 부터 애플릿을 적당히 끝마치는 기능이 약하다.
- 항상 init( ) 메쏘드를 호출하므로 가능한한 init( ) 메쏘드를 작고 빠르게 유지해야 한다.

Ⅵ. 결어

현재 폭발적으로 증가하는 인터네트상의 호스트의 수는 웹을 브라우징할 수 있는 Mosaic의 등장에 기인한다. 특히 이를 개량한 네트스케이프는 웹 선풍을 일으키는데 중요한 계기가 되었다. 무엇보다도 웹을 구축할때 사용되는 HTML 문서를 누구나 쉽게 작성할 수 있다는 것도 중요한 요소이다. 이러한 최근 상황에서 선 마이크로시스템즈(사)는 Java라는 새로운 프로그래밍 환경을 개발하였는데 차세대 인터네트 표준언어로 채택될 것이다. 무엇보다도 바이트코드 개념의 도입에 따른 코드의 독립성과 다운로딩에 의한 무구현(no implementation)을 실현할 수 있다는 것이 큰 장점이다. Java 클래스를 정의한 API 패키지의 양이 바인더 3개 분량으로 많기 때문에 본고에서는 각각에 대한 사항을 자세히 기술하지는 않았다. 그러나, 그 중요개념과 탄생의 숨겨진 이야기, 특성, 개발도구, 구체적인 예, native 메쏘드, 제약사항들을 살펴봄으로써 Java의 전체 흐름을 파악할 수 있다. Java의 응용 또한 직접 웹 브라우저를 통해서 쉽게 경험할 수 있고 더 현실적이기 때문에 본 고에서는 기술하지 않았다.

국내외에서 불고 있는 인터네트와 웹과 Java를 무작정 수용하기 보다는 이를 내 것으로 소화하고 창조할 수 있는 국내 연구 분위가 시급한 실정이다. Java의 응용은 무궁무진하지만 HTML과 같이 일반 사용자에게는 쉽지않고 최소한 C와 C++ 의 경험이 요구되고, 객체지향이라는 또 다른 사고방식을 필요로 하기 때문에 처음에 숙달 과정이 요구된다.

현재 ETRI의 정보통신표준연구센터의 Java 팀은 MBone상에서 그룹회의를 위한 애플릿(네트스케이프에서 동작)을 연구중이다.

Java 관련 자료, 프로그램, 애플릿, JDK1.0등의 정보는 http://java.sun.com에서 상당히 얻을 수 있고, 특히 에니메이션, 예술, 통신, 교육, 재정, 게임, 그래픽, 네트워크, Java 프로그래밍을 포함한 각종 Java 프로그램은 http://www.gamelan.com에서 얻을 수 있다. 문의 사항은 sunchoi@pec.etri.re.kr 또는 국내 Java 개발그룹인 java@voyager.kti.co.kr에게 연락하면 된다.

[참고문헌]

[1] Sun Microsystems, Writing Java Programs, 1995.
[2] Sun Microsystems, The Java Language Environement, Oct, 1995.
[3] Sun Microsystems, The Java Language: A White Paper, 1994.
[4] Sun Microsystems, Java API Packages.
[5] Sun Microsystems, The Java Virtual Machine.
[6] Sun Microsystems, Java(tm): Java 홈 페이지 (Programming for the Internet, http://java.sun.com).
[7] Netscape Communications Corporation, NETSCAPE JAVASCRIPT.
[8] M. OConnell, Java: The inside story, SunWorld Online, July 1995.

최 선 완

- 1984. 2 홍익대학교 전자계산학과(학사)
- 1986. 2 한국과학기술원 전산학과(석사)
- 1996. 2 한국과학기술원 전산학과(박사)
- 1986. 2 ~ 1996. 2 한국전자통신연구소 선임연구원
- 1996. 3 ~ 현재 안양대학교 정보통신공학과 전임강사
- 1996. 1 ~ 1996. 2 첨단종합유통시스템 기술연구과제책임자
- 관심분야 : 인터넷에서의 공동작업, CALS, 프로토콜공학