<http://sinuk.egloos.com/2676307>

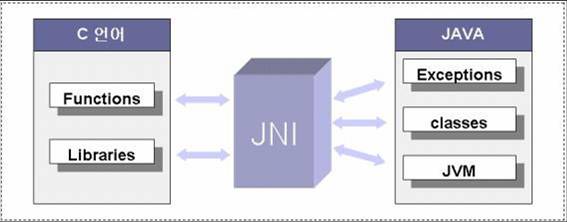
**1. JNI (Java Native Interface) 란 ?**

- 자바가 다른 언어로 만들어진 어플리케이션과 상호 작용할 수 있는 인터페이스를 제공한다.

- 자바가상머신(JVM)이 원시 메소드(native method)를 적재(locate)하고 수행(invoke)할 수 있도록 한다

- JNI가 자바가상머신내에 포함됨으로써, 자바가상머신이 호스트 운영체제상의 입출력, 그래픽스, 네트워킹, 그리고 스레드와 같은 기능들을 작동하기 위한 로컬시스템호출(local system calls)을 수행할 수 있도록 한다.

**\* 쉽게 말해 Java와 다른 언어를 연동하는 솔루션입니다.**



[그림1] C로 만들어진 Library와 JAVA를 연결해주는 JNI

**2. Why do you need JNI ?**

자바 네이티브 메쏘드(Java Native method, 이하 JNI)는 다른 언어로 작성된 코드를 자바에서 호출하도록 만들어진 규약이다. 현재는 C/C++에 대한 호출만을 정확하게 지원한다. 어떻게 보면 JNI는 자바가 만들어진 철학과 정반대되는 것이다.

그러나. **Java에도 한계**가 있다.

**1. 속도 문제가 있는 계산 루틴**   
 > 자바가 Native Code(플랫폼에 종속적인 기계어 코드)에 비해 느리다.

**2. 자바에서 하드웨어 제어**

**3. 자바에서 지원되지 않은 특정 운영체제 서비스**   
 > 자바의 클래스 라이브러리는 방대하고 다양한 서비스를 제공하지만, 특정 플랫폼에서 제공하는 고유의 서비스의 기능을 모두 포함할 수는 없다. 특히, 특수한 목적으로 제작된 하드웨어를 자바에서 제어해야 할 필요가 있다고 한다면, 자바만으로 해결하기는 힘들다.

**4. 기존의 프로그램에서 자바가 제공하는 서비스를 이용**   
 > 기존에 작성된 프로그램이나 기존의 시스템(legacy)과의 연계 문제

∴ JNI를 써서 해결해보자.

**3. C를 이용한 JNI 예제**

VC++을 이용해 C문법으로 작성되어 만들어진 DLL을 로딩하여 Java에서 사용해보겠습니다.

**1단계 : Native Method를 선언하는 자바 클래스 작성  
2단계 : 1단계에서 작성한 클래스 컴파일  
3단계 : javah를 사용해서 Native Method가 사용할 헤더 파일 생성  
4단계 : C언어로 Native Method 실제 구현  
5단계 : C 코드와 헤더 파일을 컴파일  
6단계 : 자바 프로그램 실행**

**1단계 : Native Method를 선언하는 자바 클래스 작성**

Java 소스 파일 : **HelloJni\_Jsource.java**

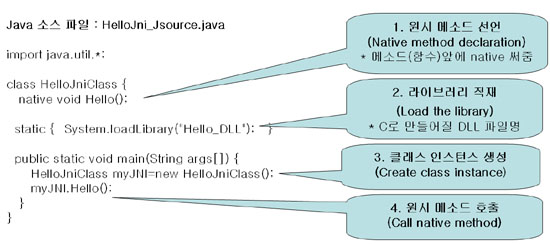
import java.util.\*;

class HelloJniClass {  
   native void Hello();

  static {  System.loadLibrary("Hello\_DLL");   }

  public static void main(String args[]) {      
      HelloJniClass myJNI=new HelloJniClass();  
      myJNI.Hello();  
   }  
}

// 아래는 좀 위의 내용 보충 그림



**2단계 : 1단계에서 작성한 클래스 컴파일**

http://tfile.nate.com/download.asp?FileID=10805099  
  
\* 컴파일시에는 일반 java 컴파일때와 마찬가지로 환경변수 셋팅이 되어 있어야 합니다.  
 -> Path가 JDK의 Javac.exe가 있는 폴더에 설정되어 있어야 합니다.

**3단계 : javah를 사용해서 Native Method가 사용할 헤더 파일 생성**

**http://tfile.nate.com/download.asp?FileID=10805100**

HelloJniClass.h을 열어보면

JNIEXPORT void JNICALL Java\_HelloJniClass\_Hello  (JNIEnv \*, jobject);  
위의 함수를 Implement만 해서 DLL을 만들면 됩니다. (4단계)

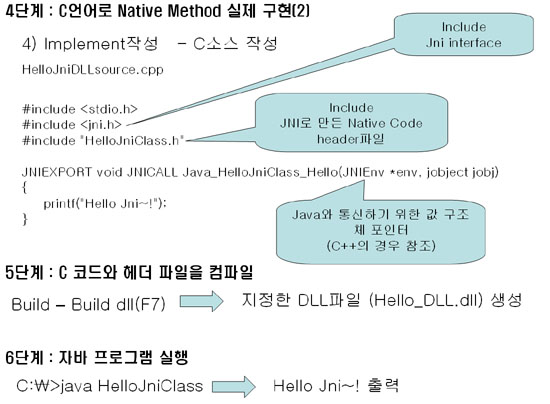
**4단계 : C언어로 Native Method 실제 구현(1)**

1) VC++ 프로젝트 만들기 : Win32용 DLL 프로젝트로 만듭니다.  
New - Projects : Win32 Dynamic-Link Library

2) Add Files Projects : HelloJniClass.h 파일 추가

3) Projects Setings(Alt+F7)  
   - Link탭에 Output file Name : 1단계의 2. 라이브러리 적재시 작성한 DLL파일명(Hello\_DLL.dll)  
   - C/C++탭 Preprocessor 카테고리의 Additional Include directories   
       JDK의 Include폴더와 Include폴더 밑의 win32폴더

          예) C:\Program Files\Java\jdk1.5.0\_03\include\,   
             C:\Program Files\Java\jdk1.5.0\_03\include\win32



**4. 값의 전달과 리턴**

**1단계 : Java 소스 파일 StringPass\_Jsource.java**  
\* 일반 자바 메쏘드 선언과 동일합니다.

class JNI\_Message {  
   native byte[] Message(String input);

  // 라이브러리 적재(Load the library)

  static {  
    System.loadLibrary("Msg\_DLL");  
  }

  public static void main(String args[]) {      
 byte buf[];

    // 클래스 인스턴스 생성(Create class instance)  
    JNI\_Message myJNI=new JNI\_Message();

    // 원시 메소드에 값을 주고 받음  
    buf = myJNI.Message("Apple");  
   
 System.out.print(buf); // 받은값 출력  
 }  
}

**2단계 : 컴파일**  
 javac StringPass\_Jsource.java

**3단계 : header파일 생성** javah JNI\_Message

**4단계 : method구현 : StringJNIDLLSource.c**

#include <stdio.h>  
#include <jni.h>  
#include <string.h>  
#include "JNI\_Message.h"

JNIEXPORT jbyteArray JNICALL Java\_JNI\_1Message\_Message (JNIEnv \* env, jobject jobj, jstring input)   
{  
    jbyteArray jb;  
    jboolean iscopy;  
    char\* buf;  
    static char outputbuf[20];  
  
    buf=(\*env)->GetStringUTFChars(env, input, &iscopy);  // 입력 String 읽어오는 함수  
    printf ("\nDLL receive Data from JAVA : %s\n",buf);   // 입력받은 내용을 출력  
    strcpy(outputbuf,"Delicious !!\n");   
    jb=(\*env)->NewStringUTF(env, outputbuf);  // 출력할 내용의 java버퍼에 output버퍼값을 셋팅  
  
   return(jb); // java버퍼 리턴  
}

(\*env)->함수명 형태로, JAVA의 메쏘드를 C에서 이용할수 있습니다.  
\* JAVA는 C로 문자열을 넘겨줄때 UTF-8형태를 사용합니다.

**5단계 : 실행**

C:\test\C\_JNI\Paramerter Pass>java JNI\_Message

DLL receive Data from JAVA : Apple  
Delicious !!

**5. KVM ? KNI ?**

KVM은 J2ME의 일부로서 작고 자원이 한정된 기계장치를 위해 설계된 소형 JVM.  
JVM에서는 JNI가 KVM의 KNI가 있다.