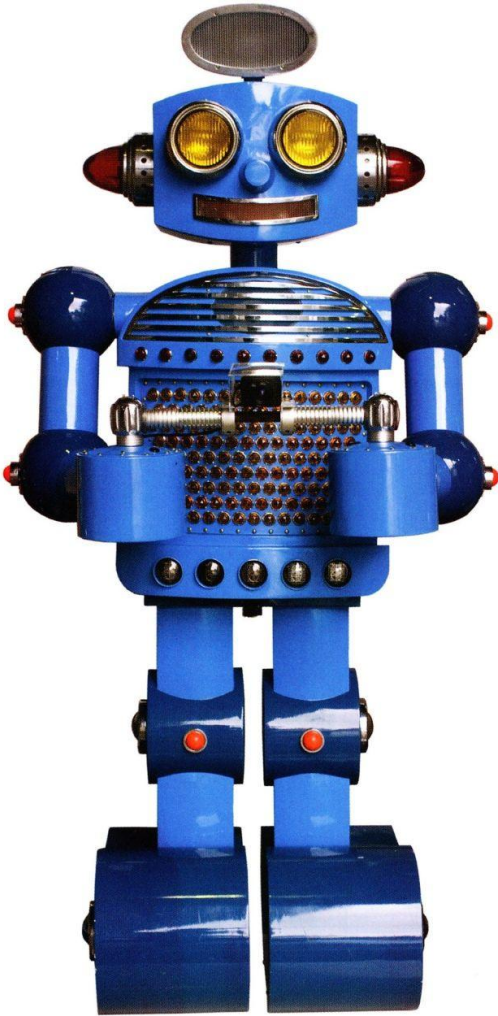


Desarrollo Nativo Android (JNI)

Desarrollo Nativo Multiplataforma





Marcos J. Ortega Morales

Nicaragüense

(Managua <- La Paz Centro <- León)

Ing. en Computación

Universidad Nacional de Ingeniería (NI)

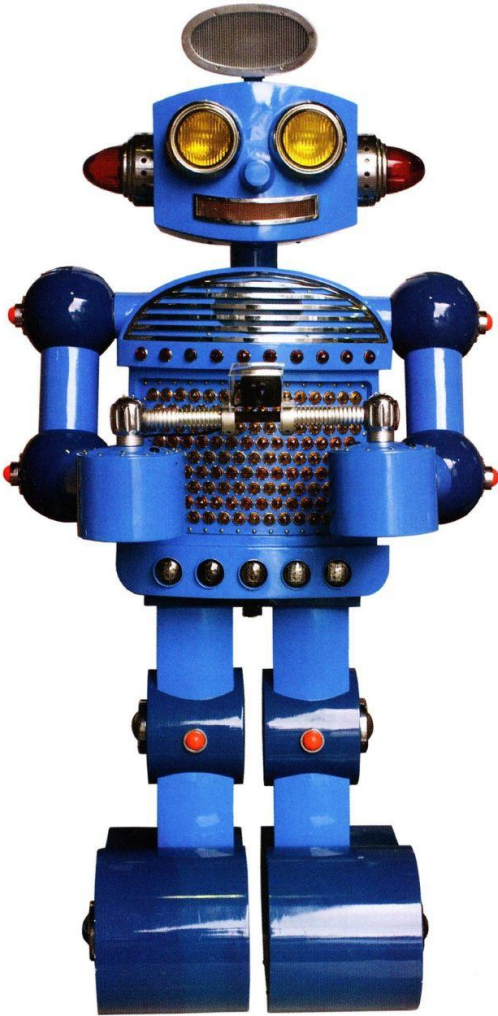
Curso Desarrollo Videojuegos

Universidad Complutense de Madrid (ES)

Gerente General y Director de Desarrollo en
Nicaragua Binary S.A.

marcosjom01@gmail.com

mortegam@nibsa.com.ni



Mi contexto

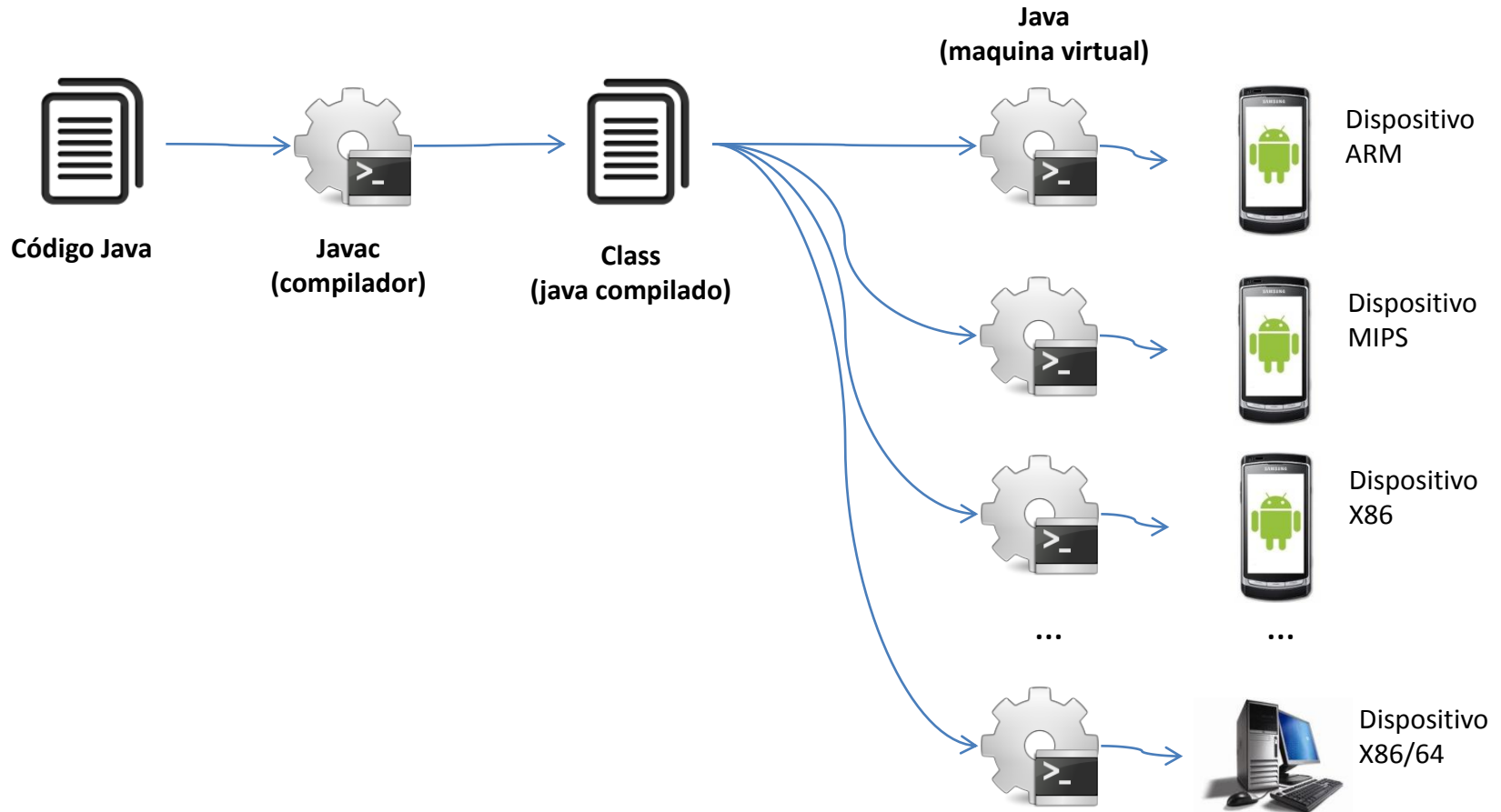
**Arquitectura de máquinas
(vista desde abajo)**

**Maximizar la reutilización de código
(el bueno, el malo y el feo)**

**Desarrollo Nativo Multiplataforma
(conocí C y me inspiré en Java)**



Generalidades Java





Generalidades Java

Ventajas

- Código mas simple.
- Compilados ejecutan en múltiples arquitecturas (en máquina virtual).

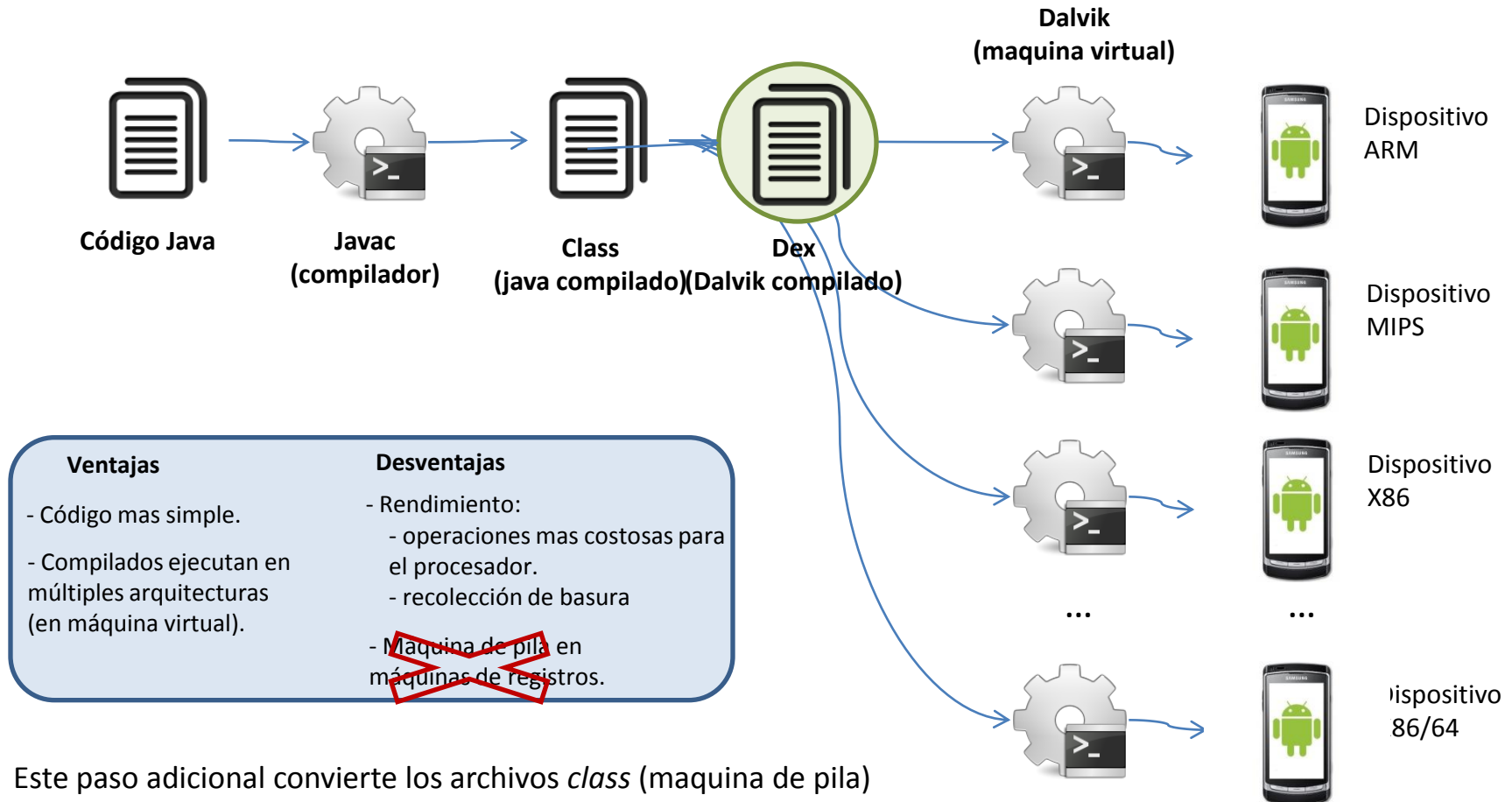
Desventajas

- Rendimiento:
 - operaciones mas costosas para el procesador.
 - recolección de basura
- Las instrucciones en los class son de máquina de pila, mientras que los procesadores generalmente son máquinas de registros.



Generalidades Java

(Android)

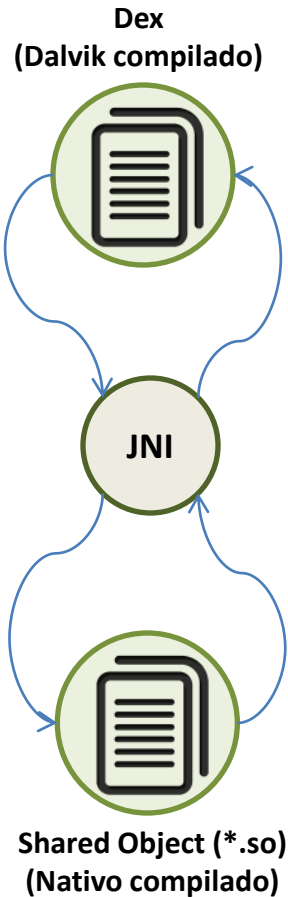


Este paso adicional convierte los archivos *class* (maquina de pila) a formato *dex* (maquina de registros), reduciendo significativamente la brecha de rendimiento.



Generalidades JNI

(Android)



Ventajas

- Conexión con compilados nativos.

Desventajas

- Existe un costo al pasar de nativo a virtual y viceversa.

Ventajas

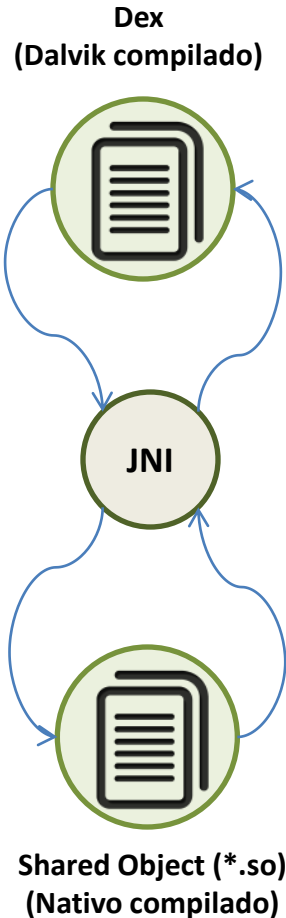
- Código más simple.
- Compilados ejecutan en múltiples arquitecturas (en máquina virtual).

Desventajas

- Rendimiento:
 - operaciones más costosas para el procesador.
 - recolección de basura
- Máquina de pila en máquinas de registros.



Generalidades JNI



Cuando conviene JNI?

Para tareas de procesamiento costoso:

- Simulación de física
- Análisis de bancos de datos
- Cálculos costosos: crypto, audio, video, etc...

Para reutilización de código nativo:

- Uso de librerías de terceros (OpenSSL, etc...)
- Migración de apps/libs nativas

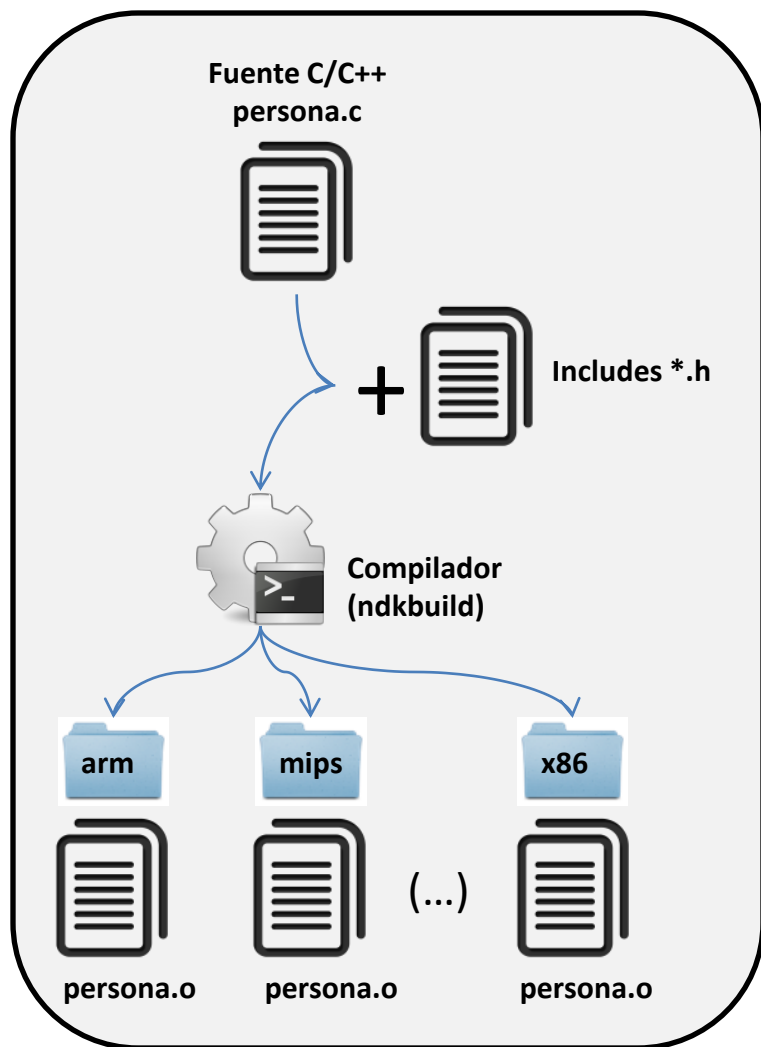
Para mayor flexibilidad:

- Acceso a las APIs de sistemas operativos.
- Acceso con privilegios distintos a los de máquina virtual.

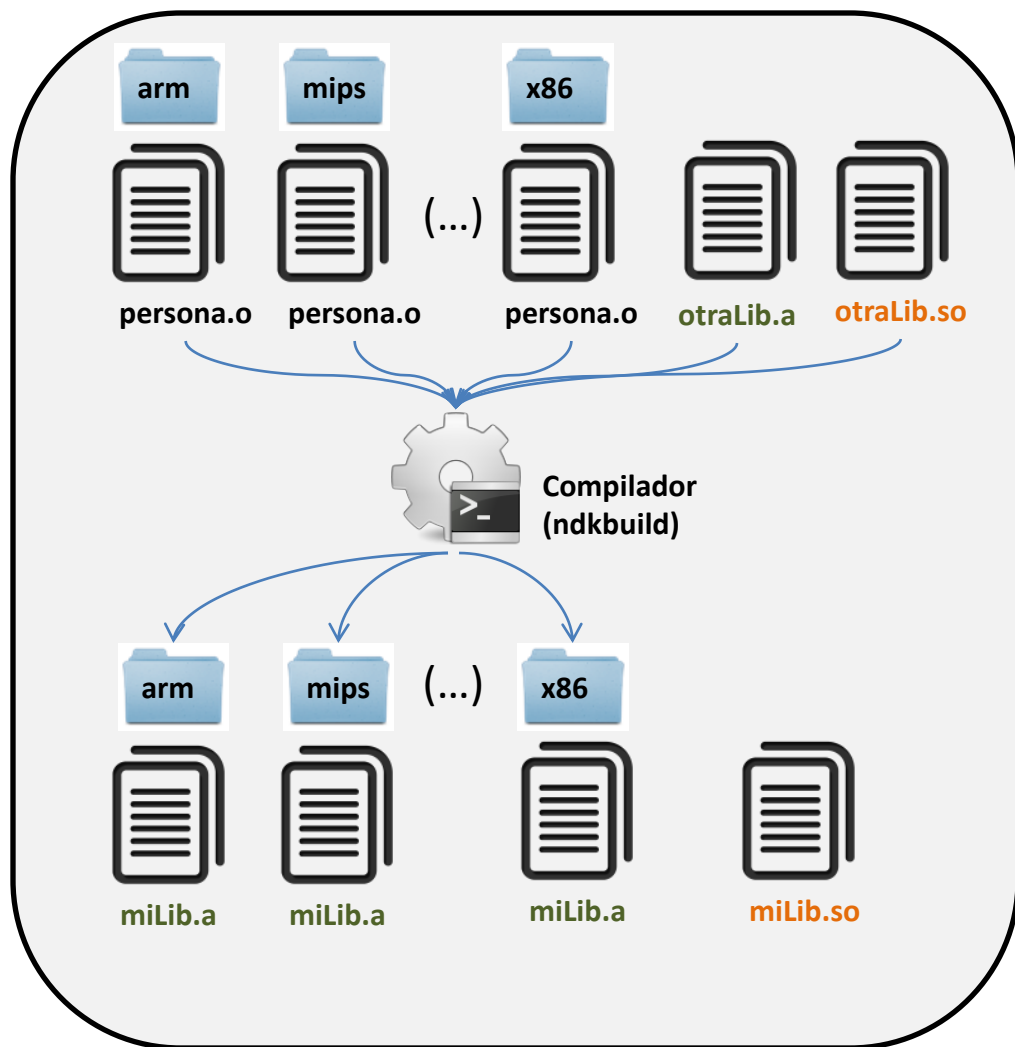


Proceso de compilación nativa

Etapa compilación

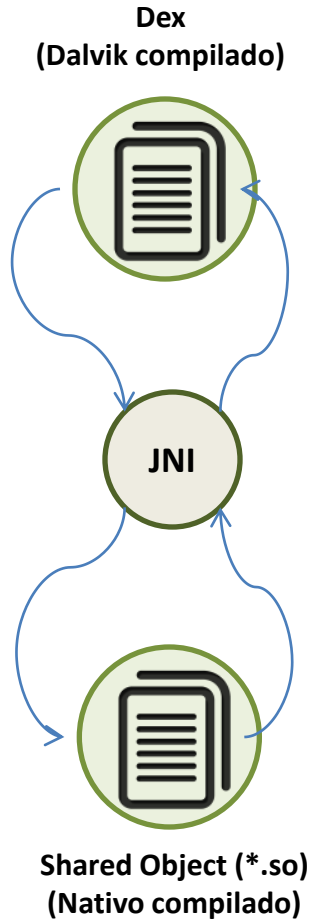


Etapa linkeo





Herramientas JNI



Android Native Development Kit (NDK)

javac

Compila un archivo “java” hacia “class”.

javah

Genera un archivo “.h” con los prototipos de métodos Nativos declarados dentro de un archivo “class”.
Esta es la base de la Interfaz Nativa (JNI).

ndkbuild

Compila, linkea y crea las librerías nativas, a partir de la Configuración definida en los archivos “jni/Android.mk” Y “jni/Application.mk”.



Caso práctico (JNI)

Vamos a desarrollar y compilar una librería para leer el contenido de un archivo dentro del paquete del App.

Herramientas a usar



Eclipse + ADT Plugin



Android NDK

Una vez que funcione para Android, haremos que el código nativo compile y funcione para Apps iOS, OSX, Windows, Blackberry 10, Linux y otros...



Caso práctico (JNI)

Librería que vamos a consumir a través de JNI:

```
//  
// LectorAssets.h  
//  
// Created by Marcos Ortega Morales on 21/06/2014.  
//  
  
#ifndef LECTORASSETS_H_INCLUDEDO  
#define LECTORASSETS_H_INCLUDEDO  
  
class LectorAssets {  
public:  
    static bool inicializar(AAssetManager* apkAssetsMgr);  
    static const char* dameContenidoAssetEnPaquete(const char* rutaVirtualArchivo);  
    static void finalizar();  
private:  
    static char* _tmpContenido;  
    static int _tmpContenidoUso;  
    static int _tmpContenidoTamano;  
};  
  
#endif
```



Caso práctico (JNI)

Paso 1:



Abrir Eclipse y crear un nuevo proyecto App para Android.

Paso 2:



Crear una clase java en la que se definirán los métodos nativos:
AppNativo.java

```
package demo.marcos.ortega.jni;

public class AppNativo {
    private native boolean nativeIniciar(Object assetManager);
    private native String nativeDameContenidoAsset(String rutaVirtual);
    private native void nativeFinalizar();
}
```



Caso práctico (JNI)

Paso 3:



Crear la carpeta “jni” dentro del proyecto.

Paso 4 (opcional):



Compilar “*AppNativo.java*” para generar el “*AppNativo.class*”

```
DemoJNI usuario$ javac -d ./bin/classes/ ./src/demo/marcos/ortega/jni/AppNativo.java
```

Paso 5 (opcional):



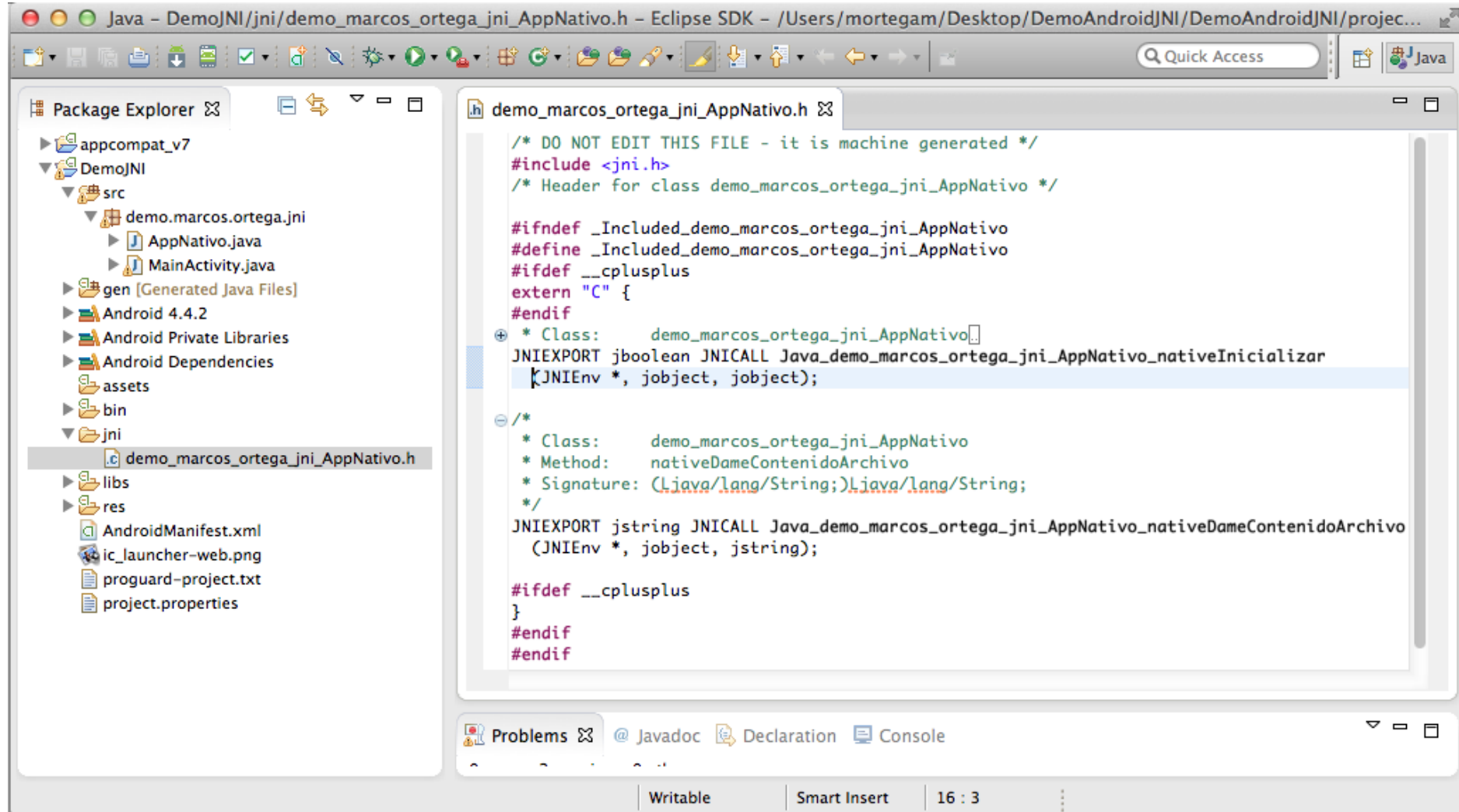
Generar el encabezado nativo a partir del “*AppNativo.class*”

```
DemoJNI usuario$ javah -d ./jni -classpath ./bin/classes/ demo.marcos.ortega.jni.AppNativo
```



Caso práctico (JNI)

El resultado hasta ahora



The screenshot shows the Eclipse IDE interface. The Package Explorer on the left displays the project structure: appcompat_v7, DemoJNI, src (containing demo.marcos.ortega.jni, AppNativo.java, and MainActivity.java), gen [Generated Java Files], Android 4.4.2, Android Private Libraries, Android Dependencies, assets, bin, jni (containing demo_marcos_ortega_jni_AppNativo.h), libs, and res. The main editor window shows the content of demo_marcos_ortega_jni_AppNativo.h, which is a machine-generated header file. The file includes <jni.h> and defines the class demo_marcos_ortega_jni_AppNativo. It contains two JNIEXPORT functions: JNIEXPORT jboolean JNICALL Java_demo_marcos_ortega_jni_AppNativo_nativeInicializar and JNIEXPORT jstring JNICALL Java_demo_marcos_ortega_jni_AppNativo_nativeDameContenidoArchivo. The file also includes conditional compilation directives for __cplusplus.

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class demo_marcos_ortega_jni_AppNativo */

#ifdef _Included_demo_marcos_ortega_jni_AppNativo
#define _Included_demo_marcos_ortega_jni_AppNativo
#ifdef __cplusplus
extern "C" {
+ * Class:      demo_marcos_ortega_jni_AppNativo
JNIEXPORT jboolean JNICALL Java_demo_marcos_ortega_jni_AppNativo_nativeInicializar
    (JNIEnv *, jobject, jobject);

- /*
 * Class:      demo_marcos_ortega_jni_AppNativo
 * Method:     nativeDameContenidoArchivo
 * Signature:  (Ljava/lang/String;)Ljava/lang/String;
 */
JNIEXPORT jstring JNICALL Java_demo_marcos_ortega_jni_AppNativo_nativeDameContenidoArchivo
    (JNIEnv *, jobject, jstring);

#ifdef __cplusplus
}
#endif
#endif
```



Caso práctico (JNI)

Paso 6:

Crear el archivo *“demo_marcos_ortega_jni_AppNativo.cpp”*

El comando “javah” ha generado el archivo de encabezado *“demo_marcos_ortega_jni_AppNativo.h”* con los prototipos de los métodos JNI de nuestra clase “AppNativo”.

(en Java)

```
package demo.marcos.ortega.jni;  
public class AppNativo {  
    private native boolean nativeInicializar(Object assetManager);  
}
```



(en JNI, C/C++)

```
JNIEXPORT jboolean JNICALL Java_demo_marcos_ortega_jni_AppNativo_nativeInicializar  
(JNIEnv *, jobject, jobject);
```

Ahora debemos generar el cuerpo de esos métodos JNI.



Caso práctico (JNI)

Paso 6 (continuación):

Contenido base para “demo_marcos_ortega_jni_AppNativo.cpp”

```
#include "demo_marcos_ortega_jni_AppNativo.h"

JNIEXPORT jboolean JNICALL Java_demo_marcos_ortega_jni_AppNativo_nativeInicializar
(JNIEnv* entorno, jobject instancia, jobject assetsManager){
    return JNI_TRUE;
}

JNIEXPORT jstring JNICALL Java_demo_marcos_ortega_jni_AppNativo_nativeDameContenidoAsset
(JNIEnv* entorno, jobject instancia, jstring rutaVirtualAsset){
    jstring jStrContenido = entorno->NewStringUTF("");
    return jStrContenido;
}

JNIEXPORT void JNICALL Java_demo_marcos_ortega_jni_AppNativo_nativeFinalizar
(JNIEnv* entorno, jobject instancia){
    //
}
```



Caso práctico (JNI)

Paso 7:

Crear los archivos de compilación (MK) dentro de la carpeta JNI.

jni/Application.mk

```
APP_ABI := all #armeabi armeabi-v7a mips x86
```

jni/Android.mk

```
NB_LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)
LOCAL_PATH      := $(NB_LOCAL_PATH)
LOCAL_MODULE    := lectorassets
LOCAL_LDLIBS    := -llog -landroid #-landroid (AssetsManager)
LOCAL_C_INCLUDES:= $(LOCAL_PATH)/../..../include/
LOCAL_SRC_FILES := demo_marcos_ortega_jni_AppNativo.cpp
include $(BUILD_SHARED_LIBRARY)
```

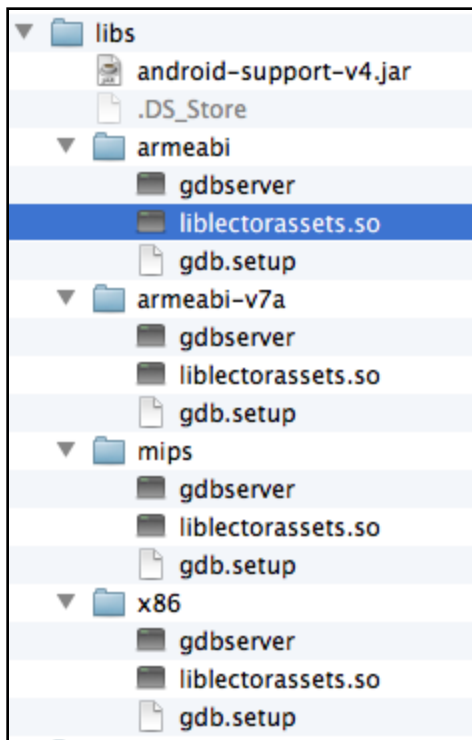


Caso práctico (JNI)

Paso 8:

Compilar la librería.

```
DemoJNI usuario$ /Applications/android-ndk-r8b/ndk-build TARGET_PLATFORM=android-9 V=1 NDK_DEBUG=1
```



El comando “ndk-build” leerá el contenido de los archivos “Android.mk” y “Application.mk”, compilará los archivos fuente según se especificó y creará las librerías estáticas y dinámicas listas para ser incluidas en el App.



Caso práctico (JNI)

Paso 9:

Automatizar la compilación de la librería nativa cada vez que se compile el proyecto Android.

jnibuild.sh

```
#!/bin/sh

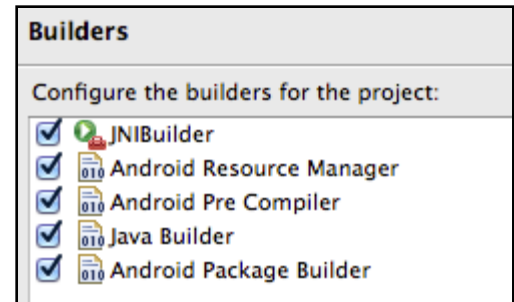
#Set the NDK path
export ANDROID_NDK_ROOT=/Applications/android-ndk-r8b

#Generate JNI interface "JAVA -> CLASS"
javac -d ./bin/classes/ ./src/demo/marcos/ortega/jni/AppNativo.java

#Generate JNI interface "CLASS -> C"
javah -d ./jni -classpath ./bin/classes/ demo.marcos.ortega.jni.AppNativo

#$ANDROID_NDK_ROOT/ndk-build clean
$ANDROID_NDK_ROOT/ndk-build TARGET_PLATFORM=android-9 V=1 NDK_DEBUG=1
```

Agregar nuevo builder:



Al agregar el script en una etapa Builder, Eclipse se encargará de compilar la librería nativa cada vez que el proyecto Android sea compilado.

Ahora podemos manipular los archivos fuente sin tener que trabajar con la consola.



Caso práctico (JNI)

Explorar proyecto

github.com/marcosjom/AndroidDemoJNI



Código nativo multiplataforma



Nativo Multiplataforma

Conocimientos sobre el funcionamiento en mas de una plataforma.

+

Reprocesador de C/C++
#include
#define
#ifdef
#ifndef
#elif
...

= Código C/C++/ObjC que compila y ejecuta en múltiples plataformas.



Nativo Multiplataforma

```
#if defined(_WIN32)
    //Código Windows
#elif defined(__ANDROID__)
    //Android
#elif defined(__QNX__)
    //QNX 4, QNX Neutrino, or BlackBerry 10 OS
#elif defined(__linux__) || defined(linux)
    //Linux
#elif (defined(__APPLE__) && defined(__MACH__))
    //MacOSX
#elif defined(__APPLE__)
    //MacOSX o iOS
#else
    //otro?
#endif
```




Nativo Multiplataforma

Explorar proyecto

github.com/nicaraguabinary/lib-nixtla-audio



Buen provecho



marcosjom01@gmail.com
mortegam@nibsa.com.ni