14장 타깃 장치로 고급 포팅하기

시작하면서

- **목차**
  - 14.1 안드로이드에서 C 프로그램 실행시키기
  - 14.2 안드로이드 NDK 설치
  - 14.3 NDK 기초 실습
  - 14.4 NDK 고급 실습

14.1 C 프로그램 실행시키기

- **안드로이드 플랫폼에서 애플리케이션**
  - 자바
  - C/C++
  - 자바 애플리케이션과 C/C++ 프로그램과의 상호 연동
- **에뮬레이터에서 apk 패키지 아닌 C/C++ 프로그램 실행**

14.1 C 프로그램 실행시키기

[실습 14-1] C로 helloworld.c 빌드, 실행

(1) vi helloworld.c

```
01 #include <stdio.h>
02 int main(int argc, char **argv)
03 {
04 printf("Hello World! ₩n");
05 return 0;
06 }
```

(2) arm-none-linux-gnueabi-gcc -static helloworld.c -o helloworld

다음은 실행파일을 만들기 위하여 컴파일 과정을 진행한다. gcc를 사용하여 애플리케이션 컴파일과 빌드를 한다. 정적(static) 옵션을 사용하여 빌드를 한다.

14.1 C 프로그램 실행시키기

- **[실습 14-1] C로 helloworld.c 빌드, 실행**

  (3) adb push helloworld data/helloworld

  리눅스에서 새로 작성한 애플리케이션 실행파일 helloworld을 윈도우의 에뮬레이터에서 실행해 볼 수 있도록 에뮬레이터로 복사한다. adb를 사용하여 에뮬레이터 파일시스템의 data/helloworld 디렉토리로 업로드 한다.

  (4) adb shell

  에뮬레이터 쉘모드로 간다.

  (5) chmod 777 data/helloworld

  helloworld 프로그램에 대한 소유권한을 실행가능 상태로 설정한다.

  (6) ./helloworld

  파일을 실행시키기 위하여 ./helloworld를 입력한다. 리눅스는 실행파일에 대한 경로정보를 현재 디렉토리가 포함하고 있지 않아서, 현재 디렉토리를 나타내는 접두사 "./"을 붙여준다. 간단하지만 안드로이드 에뮬레이터에서 리눅스 C 프로그램이 동작되는 것을 확인할 수 있다.

  (7) Hello World !

  실행결과 Hello World ! 가 화면에 나타나는 것을 확인한다.

14.1 C 프로그램 실행시키기

[실습 14-2] C로 hellodynamic.c 빌드, 실행

(1) <실습 14-1>처럼 hellodynamic.c를 작성한다.

(2) 에뮬레이터 /system/lib 파일을 D:₩android₩system₩lib으로 추출

에뮬레이터에서 동적 라이브러리 파일 추출 실습 내용

```
adb pull /system/lib/libdl.so  D:\android\system\lib
adb pull /system/lib/libthread_db.so   D:\android\system\lib
adb pull /system/lib/libc.so   D:\android\system\lib
adb pull /system/lib/libm.so   D:\android\system\lib
adb pull /system/lib/libGLES_CM.so   D:\android\system\lib
adb pull /system/lib/libssl.so   D:\android\system\lib
...
adb pull /system/lib/libhardware.so  D:\android\system\lib
adb pull /system/lib/libsqlite.so  D:\android\system\lib
```

14.1 C 프로그램 실행시키기

- **[실습 14-2] C로 hellodynamic.c 빌드, 실행**

  (3) arm-none-linux-gnueabi-gcc -c hellodynamic.c -o hellodynamic.o
  hellodynamic.c를 컴파일한다. -c 옵션으로 컴파일만 수행한다.

  (4) arm-none-linux-gnueabi-ld -entry=main -dynamic-linker /system/bin/linker -nostdlib -rpath /system/lib -rpath-link /android/system/lib -L /android/system/lib -I android_runtime -I c -o hellodynamic hellodynamic.o

  hellodynamic.c 프로그램에 대한 라이브러리를 동적으로 연결하여 빌드하기 위하여 arm-none-linux-gnueabi-ld를 사용한다.

  (5) SDK ₩tools₩adb push hellodynamic /data/
  리눅스 서버에서 동적으로 빌드한 hellodynamic 파일을 윈도우 기반의 SDK ₩tools 디렉터리로 복사한다.

14.1 C 프로그램 실행시키기

- **[실습 14-2] C로 hellodynamic.c 빌드, 실행**

  (6) # adb shell; # cd data; # ls -l

  윈도우에서 adb 셸 모드로 들어간 후, 에뮬레이터 /data 디렉터리로 이동한다. 파일 목록을 ls -l로 검사한다.

  2504 hellodynamic

  568231 helloworld

  (7) ./helloworld

  실행 결과로'Hello World!'를 갖는다.

  (8) ./hellodynamic

  실행 결과로

  'Hello World!'

  [1] Killed ./hellodynamic

14.2 안드로이드 NDK 설치

- **Java : JNI (Java Native Interface)**
- **안드로이드 : NDK (Native Development Kit)**
- Native method ? 안드로이드에서 사용할 수 있도록 C/C++로 만들어진 라이브러리

- 안드로이드에서는 NDK 는 C/C++로 작성된 코드를 공유 라이브러리 형태로 만들어 안드로이드 애플리케이션이 사용할 수 있도록 하는 역할

[그림 14-1] NDK 역할

14.2 안드로이드 NDK 설치

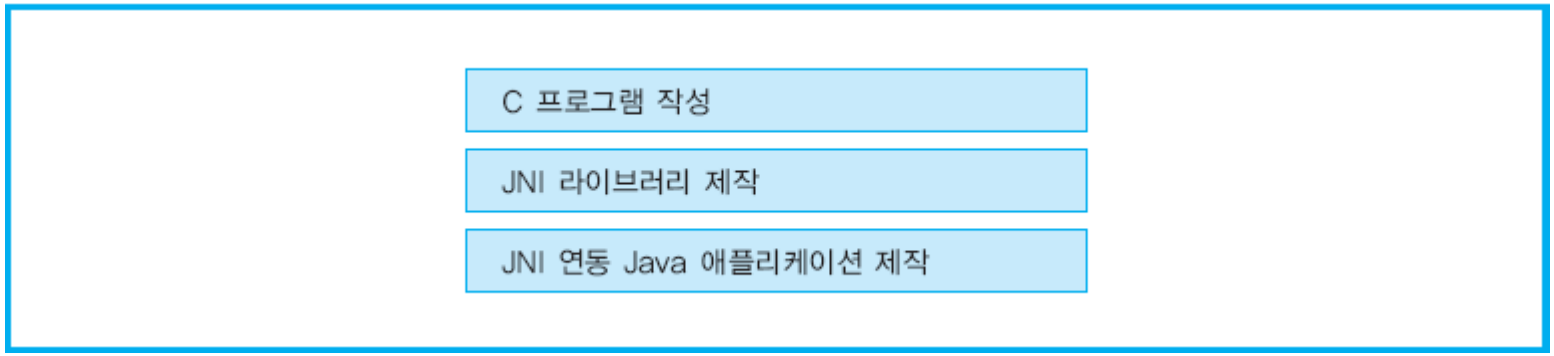
- **NDK 는 C/C++로 개발된 코드를 네이티브 라이브러리 형태로 만들어 Java에서 사용할 수 있도록 해주는 도구와 빌드 파일을 포함**
- **네이티브 API를 위하여 다음과 같은 헤더파일을 제공한다.**

  - libc(C 라이브러리) 헤더
  - libm(Math 라이브러리) 헤더
  - libz(Zlib 압축) 헤더
  - liblog(안드로이드 로깅) 헤더
  - JNI 인터페이스 헤더
  - C++ 지원을 위한 헤더

14.2 안드로이드 NDK 설치

- 안드로이드 애플리케이션 패키지 (apk)에 네이티브 라이브러리 삽입 가능
- 안드로이드 NDK 버전 r1- r4
- NDK 에는 효율적 프로그램 위한 빌드시스템 제공

- NDKK를 사용하여 C/C++ 프로그램을 안드로이드 플랫폼에서 자바 애플리케이션과 연동하여 사용하게 하는 과정 그림[14-2]



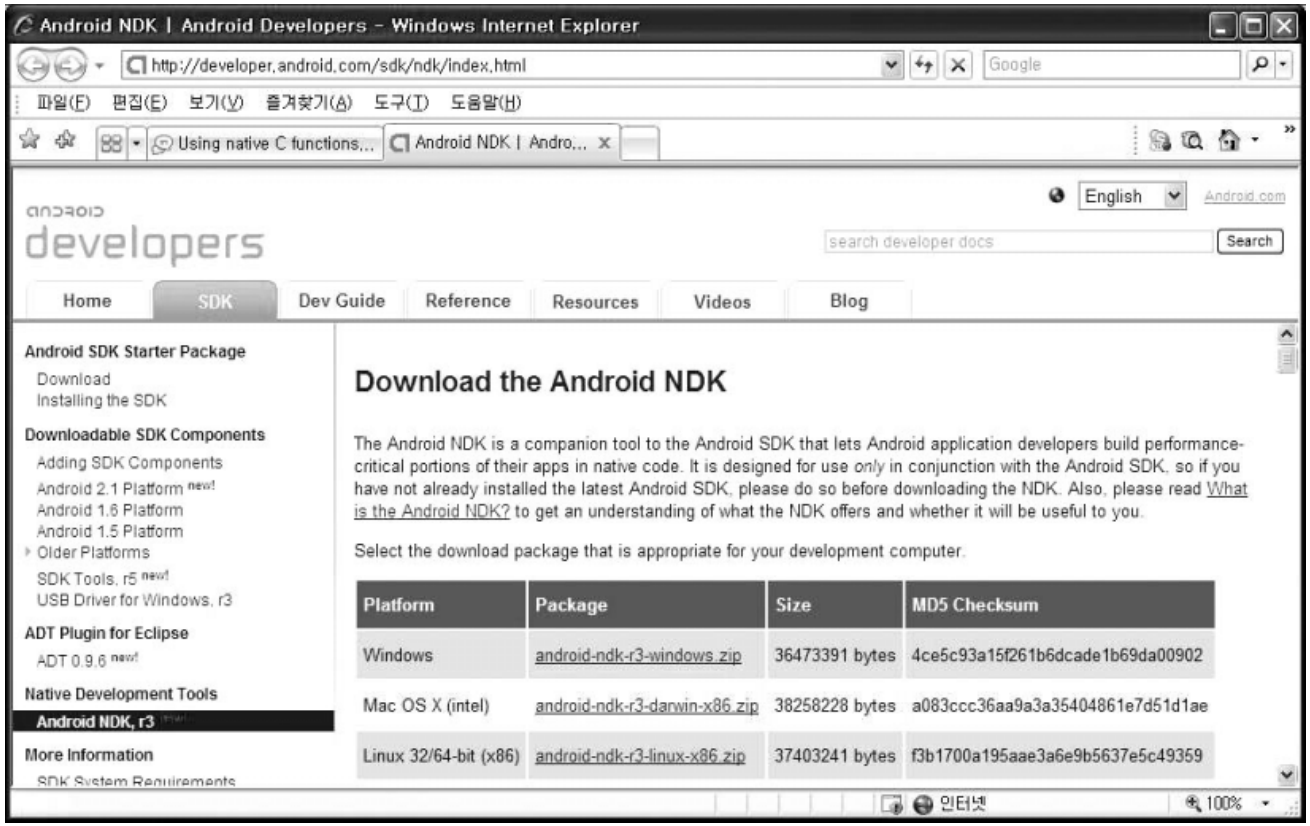
[그림 14-2] NDK 개발 절차

14.2 안드로이드 NDK 설치

- **[실습 14-3] NDK 설치하기**

  (1) http://developer.android.com/sdk/ndk/index.html

  NDK 소스를 다운로드받기 위하여 NDK 사이트 http://developer.android.com/sdk/ndk/index.html을 방문한다. 그리고 [그림 14-3]처럼 androidndk-r3-linux-x86.zip 파일을 다운로드 받는다.



[그림 14-3]
NDK 다운로드하기

14.2 안드로이드 NDK 설치

- **[실습 14-3] NDK 설치하기**

(2) # wget http://d1.google.com/android/ndk/android-ndk-r3-linux-x86.zip

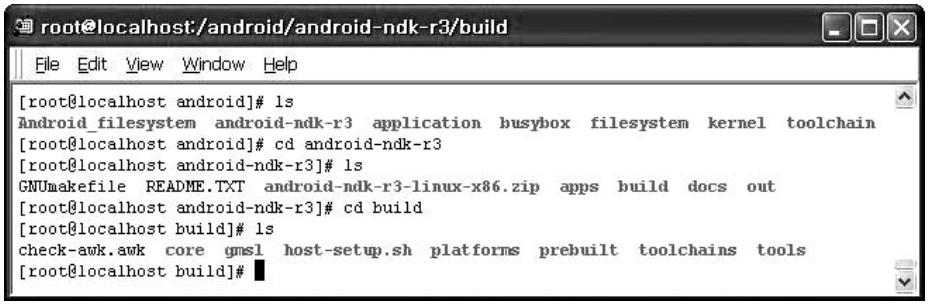
(1) 방법 대신에 wget을 사용하여 NDK 소스를 다운로드받을 수도 있다. 리눅스 서버에 /android/android-ndk-r3 디렉터리를 생성하고 리눅스용 안드로이드 NDK를 다운로드한다.

(3) unzip android-ndk-r3-linux-x86.zip

NDK 파일을 압축 해제한다.

(4) cd /android-ndk-r3

/android-ndk-r3 디렉터리에 /apps, /build, /docs, /out, GNUmakefile, README.txt 디렉터리와 파일들이 포함되어 있는 것을 [그림 14-4]처럼 확인한다. /apps 디렉터리에는 안드로이드 애플리케이션이 들어간다. NDK 예제를 빌드할 경우 해당 애플리케이션의 /libs 디렉터리에 컴파일된 라이브러리가 저장된다.



[그림 14-4] android-ndk-r3 디렉터리 내용

14.2 안드로이드 NDK 설치

- **[실습 14-3] NDK 설치하기**

(5) cd build

(6) vi host-setup.sh; cd ..

〈소스 14-1〉 host-setup.sh

```
-------------다음과 같이 수정한다--------------------
116 fi
117 local result
118 result='echo "" | $executable -f build/check-awk.awk'
119 if [ "$result" = "Pass"] ; then
120     AWK="$1"
121 fi
122 log2 "   Check $result"
------------ 저장 후 종료 -------------------------
```

(7) ./build/host-setup.sh

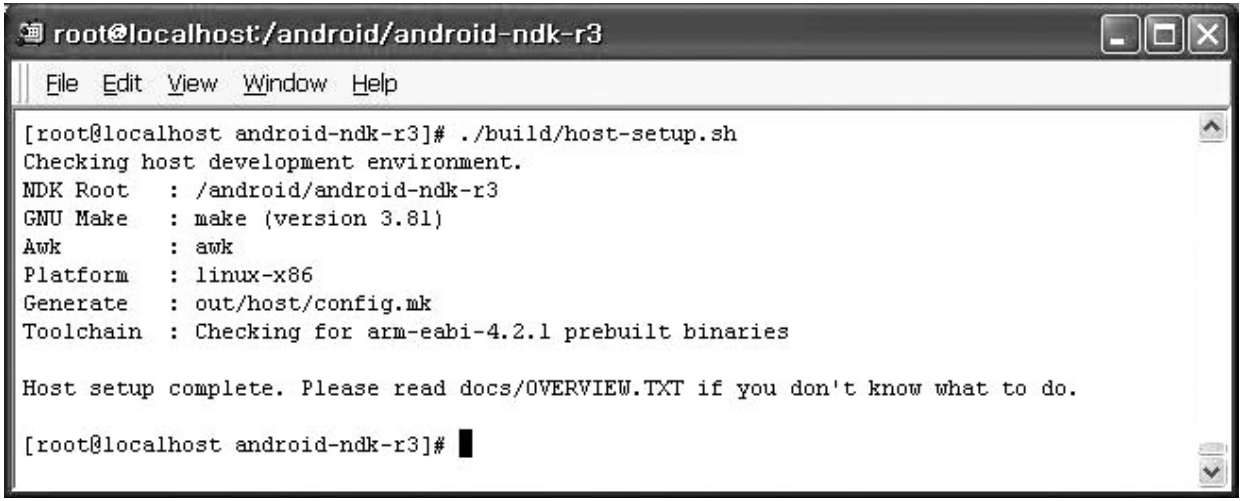
host-setup.sh를 실행해서 [그림 14-5]처럼 결과가 나타나면 NDK 설치가 완성된 것이다. 참고로 /build에서 실행하면 안 된다.

14.2 안드로이드 NDK 설치

- **[실습 14-3] NDK 설치하기**

  (7) ./build/host-setup.sh

  host-setup.sh를 실행해서 [그림 14-5]처럼 결과가 나타나면 NDK 설치가 완성된 것이다. 참고로 /build에서 실행하면 안 된다.



[그림 14-5] /build/host-setup.sh 실행

14.3 NDK 기초 실습

[실습 14-4] hellojni 애플리케이션

(1) vi hello-jni.c

NDK 설치가 정상적으로 되었는지 확인하기 위해 NDK에서 제공하는 샘플 프로그램 중에 hello-jni.c를 빌드하여 실행해본다.

hello-jni.c 파일은 단순히"Hello world, hellojni!"라는 텍스트를 출력하는 파일이다.

〈소스 14-2〉 hello-jni.c

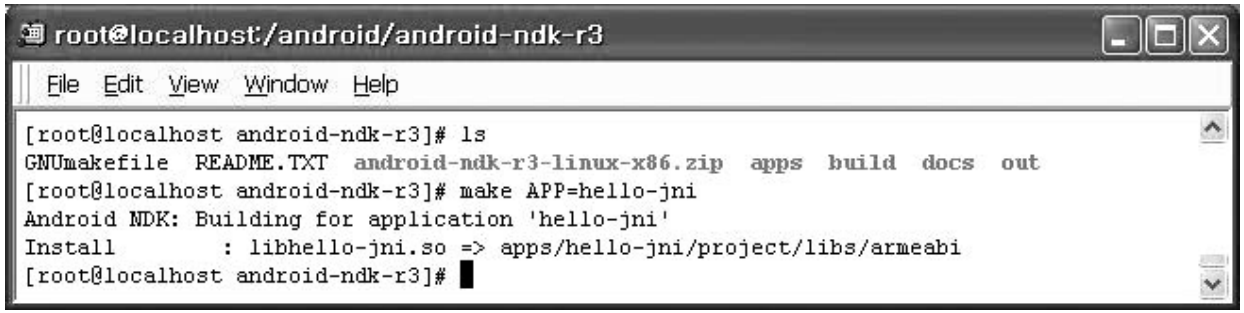
```
01 #include <string.h>
02 #include <jni.h>
03 jstring
04 Java_com_example_hellojni_HelloJni_stringFromJNI( JNIEnv* env, jobject thiz )
05 {
06 return (*env)->NewStringUTF(env, "Hello from JNI !");
07 }
```

14.3 NDK 기초 실습

- **[실습 14-4] hellojni 애플리케이션**

  (2) make APP=hello-jni

  NDK의 루트 위치에서‘make APP=hello-jni’명령어를 입력하면 [그림 14-6]처럼 된다. hello-jni.c 파일이 Android.mk 파일에 의해 libhello-jni.so 라는 라이브러리로 생성되고 helloJni.java 애플리케이션에서는 생성된 라이브러리를 참조해 구동하게 된다.



[그림 14-6] 빌드하기

14.3 NDK 기초 실습

- **[실습 14-4] hellojni 애플리케이션**

  (3) cd /armeabi; ls

  ls 명령으로 hello-jni 빌드 결과 libhello-jni.so 동적 라이브러리가 생성된 것을 [그림 14-7]처럼 확인한다.

[그림 14-7] libhello-jni.so

14.3 NDK 기초 실습

- **[실습 14-4] hellojni 애플리케이션**

  (4) HelloJni.java 파일을 작성한다.

  NDK에서 빌드된 라이브러리는 안드로이드 프로그램에서 바로 사용할 수 없다. (3)에서 확인한 libhello-jni.so를 사용하는 안드로이드 애플리케이션 프로그램을 작성해야 한다. 리눅스 서버₩android₩android-ndk-r3₩apps-hellojni₩project ₩src₩com₩example₩hellojni 경로에 있는 HelloJni.java 파일을 확인해본다. 다음 <소스 14-3>은 HelloJni.java 파일의 내용이다.

〈소스 14-3〉 HelloJni.java

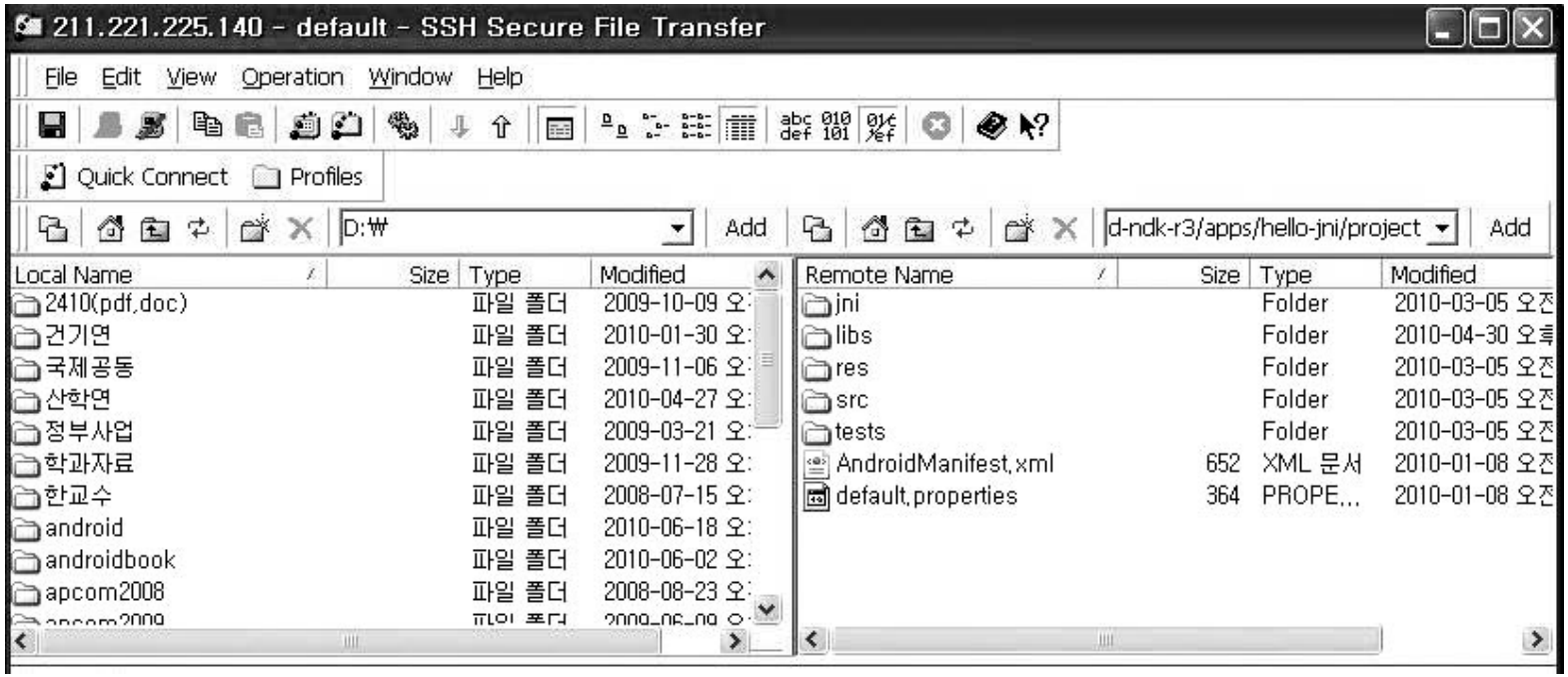
```
package com.example.hellojni;
import android.app.Activity;
import android.widget.TextView;
import android.os.Bundle;

public class HelloJni extends Activity
{
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState)
{
super.onCreate(savedInstanceState);
TextView tv = new TextView(this);
tv.setText( stringFromJNI() );
setContentView(tv);
}
public native String stringFromJNI();
public native String unimplementedStringFromJNI();
static {
System.loadLibrary("hello-jni");
}
}
```

14.3 NDK 기초 실습

- **[실습 14-4] hellojni 애플리케이션**

  (5) 리눅스 서버의 ₩android₩android-ndk-r3₩apps₩hello-jni 내용을 에뮬레이터가 동작하는 윈도우 PC D:₩android₩ndk₩hello-jni로 전송한다. [그림 14-8]처럼 진행한다.



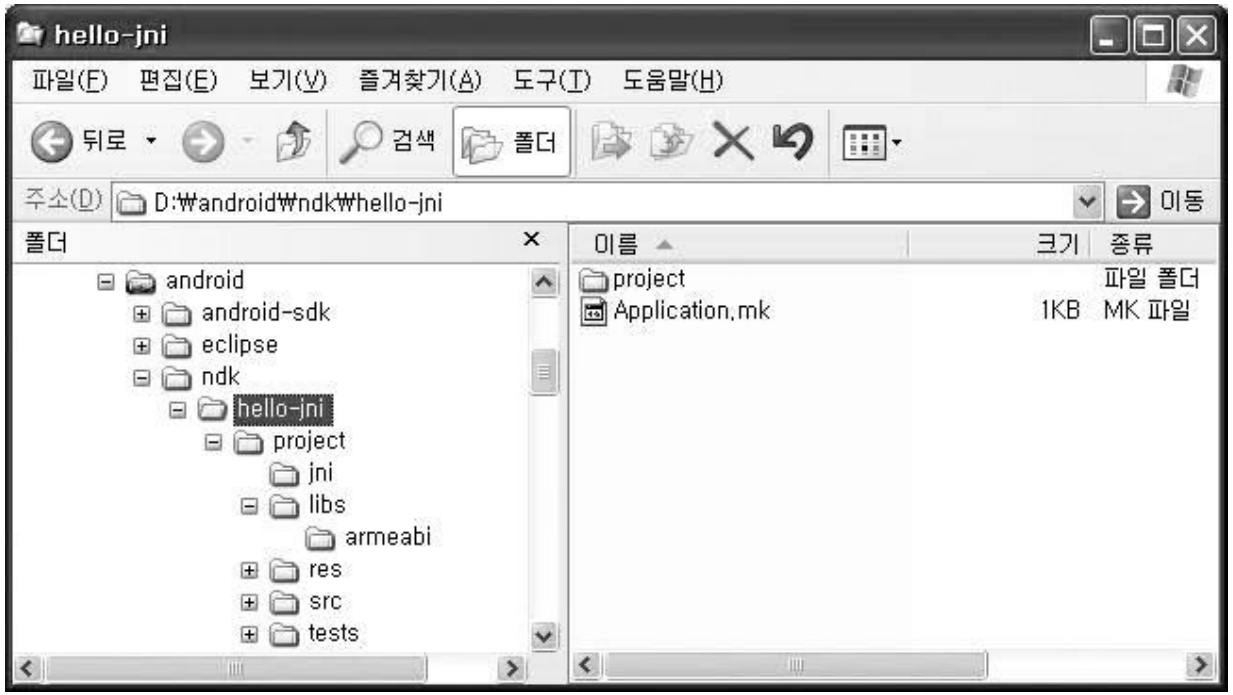
[그림 14-8] ₩android₩android-ndk-r3₩apps₩hello-jni 전송

14.3 NDK 기초 실습

- **[실습 14-4] hellojni 애플리케이션**

  (6) 윈도우 PC에 전송된 프로젝트를 확인한다.

  윈도우 PC에 전송된 프로젝트가 [그림 14-9]처럼 확인된다.



[그림 14-9] 윈도우에 android₩ndk₩로 전송된 project

14.3 NDK 기초 실습

- **[실습 14-4] hellojni 애플리케이션**

  (7) 이클립스에서 새로운 프로젝트 생성하기

  이클립스에서 새로운 프로젝트를 생성한다. 이클립스를 구동하고, File → New → Project → Next를 선택하면 새로운 안드로이드 프로젝트 생성 입력창이 나타난다.

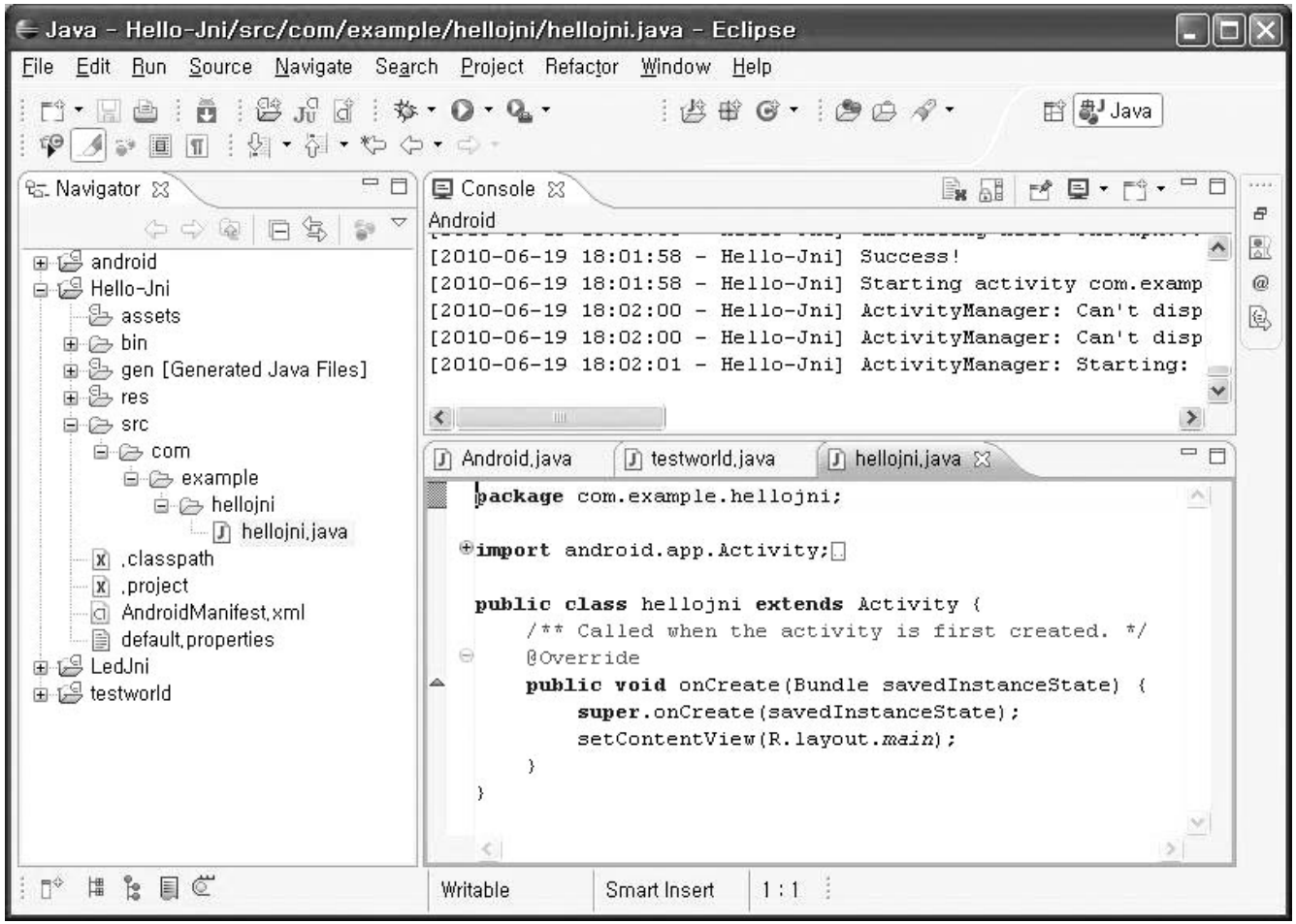
[그림 14-10] 안드로이드 프로젝트 생성

14.3 NDK 기초 실습

- **[실습 14-4] hellojni 애플리케이션**

  (7) 이클립스에서 새로운 프로젝트 생성하기 (계속)

  [그림 14-11]처럼 이클립스에 Hello-Jni 프로젝트가추가된 것을 확인할 수 있다.



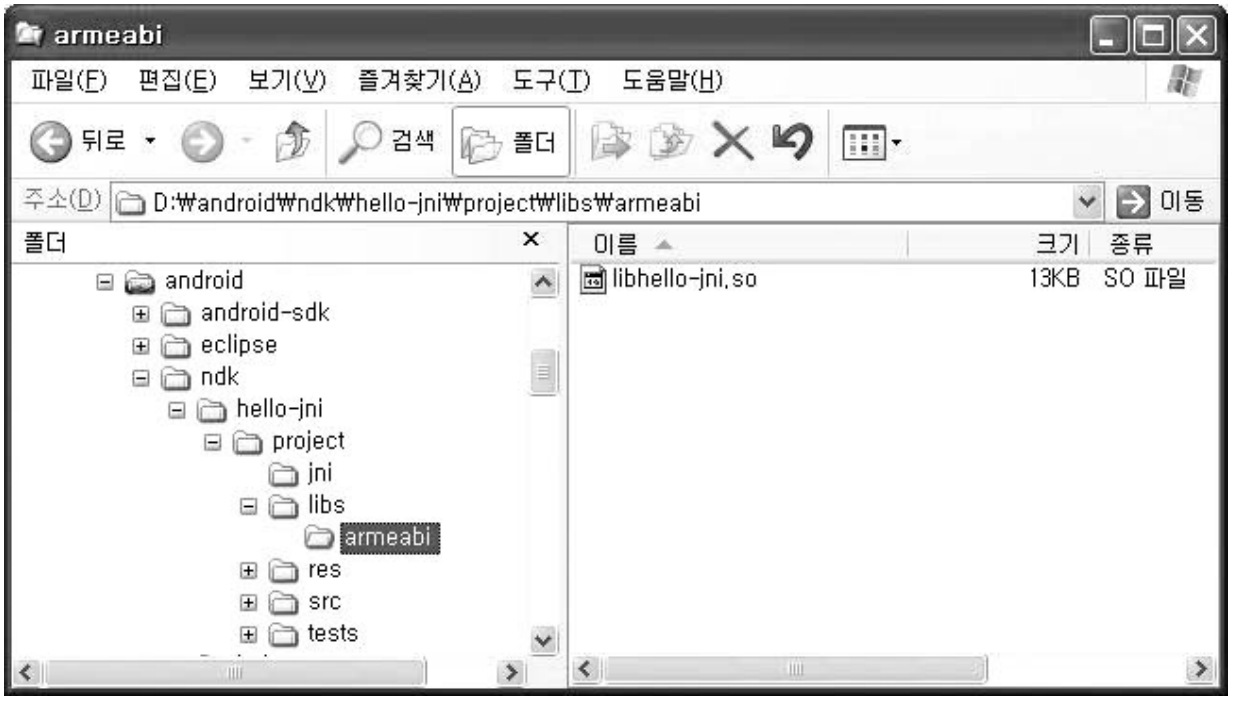
[그림 14-11] 이클립스에 생성된 프로젝트 보기

14.3 NDK 기초 실습

[실습 14-4] hellojni 애플리케이션

(8) JNI 라이브러리 추가

다음은 리눅스 서버에서 NDK로 작성한 JNI 라이브러리 ₩libs₩armeabi₩libhellojni.so를 이클립스의 프로젝트에 포함시켜야 한다. 라이브러리 ₩libs 디렉터리 전체를 [그림 14-12]처럼 D:₩android₩ndk₩project₩libs로 이동시킨다.



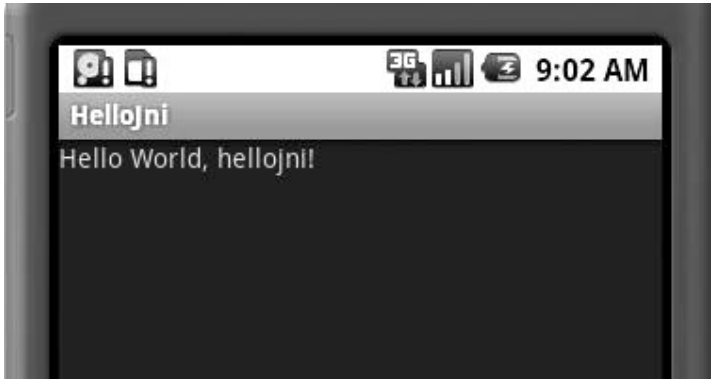
[그림 14-12] 라이브러리 추가하기

14.3 NDK 기초 실습

- **[실습 14-4] hellojni 애플리케이션**

  (9) 애플리케이션에서 라이브러리 실행하기

  이클립스 화면에서 Hello-Jni 프로젝트를 빌드하기 위하여‘RUN’한다. 에뮬레이터가 구동되면서‘Hello world, hellojni!’라는 텍스트가 [그림 14-13]처럼 출력된다.



[그림 14-13] Hello-Jni 프로젝트 실행 결과

14.4 NDK 고급 실습

- **LED 하드웨어**
  ARM GPIO 포트에 연결
  3 개의 LED
  물리주소에 비트 값 출력으로 제어

[그림 14-14] 타깃 보드의 3개의 LED 위치

14.4 NDK 고급 실습

- **LED 디바이스 드라이버**

  드라이버를 안드로이드 커널에 모듈로서 등록

  Open, read, write, module_init

  물리주소, 가상주소 매핑

드라이버 소스 led.c

```
/*
 * android device sv210 led
 *
 * by newboder
 *
 */
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/interrupt.h>
#include <linux/input.h>
#include <linux/platform_device.h>
#include <linux/miscdevice.h>
#include <linux/device.h>
#include <linux/delay.h>
#include <linux/fs.h>
#include <linux/poll.h>
#include <linux/types.h>
#include <linux/ioport.h>
#include <asm/io.h>
#include <asm/delay.h>
#include <asm/irq.h>
#include <mach/map.h>
#include <mach/gpio.h>
#include <asm/gpio.h>
#include <mach/gpio-bank.h>
#include <mach/regs-gpio.h>
#define DEVICE_NAME "sv210_led"
static int sv210_led_ioctl(struct inode *inode, struct file *file, unsigned int
cmd, unsigned long arg)
{   return 0;
}static int sv210_led_open(struct inode *minode, struct file *mfile)
{   return 0;
}static int sv210_led_release(struct inode *minode, struct file *mfile)
{   return 0;
```

14.4 NDK 고급 실습

[실습 14-5] JNI 라이브러리 생성 및 다운로드

(1) vi led_test.c

LED 드라이버를 테스트할 수 있는 프로그램 led_test.c를 JNI 라이브러리로 인터페이스를 만드는 작업을 진행한다.

JNI 프로그램 /android-ndk-r3/app/led_test/project/jni/led_test.c

```
#include <stdio.h>
#include <fcntl.h>
#include <jni.h>
jint Java_com_hybus_led_Led_LEDTest (JNIEnv*  env, jobject  thiz, jint data)
{    return main(data);
 }
#define led_dev "/dev/sv210_led"
int main(data)
{    int dev;
    int buff;
    dev = open(led_dev, O_RDWR);
            buff = data;
            write(dev,&buff, 2);
    close(dev);
    return 0;
}
```

위와 같이 선언하면 안드로이드 애플리케이션의 패키지 이름은 com_hybus_led가 된다.

14.4 NDK 고급 실습

- **[실습 14-5] JNI 라이브러리 생성 및 다운로드**

(2) make clean APP=led_test

앞에서 알아본 led_test.c 파일을 컴파일해보도록 한다. 컴파일을 하기 위해서는 리눅스 서버에 android-NDK가 설치되어 있어야 한다. LED 제어 예제에 대한 JNI app/led_test/project/jni/led_test.c 소스를 컴파일하기 전에 make clean 명령어를 통하여 기존에 있던 LED 제어 라이브러리 파일을 삭제한다.

(3) make APP=led_test

그리고 다시 make 명령으로 새롭게 [그림 14-15]처럼 컴파일한다.

[그림 14-15] JNI 라이브러리 컴파일

14.4 NDK 고급 실습

- **[실습 14-5] JNI 라이브러리 생성 및 다운로드**

  (4) cd app/led_test/project/armeabi; ls

  컴파일된 라이브러리 파일은 app/led_test/project/libs/armeabi/로 이동하여 확인해보면 앞에서 컴파일한 LED 제어 라이브러리 파일이 생성된 것을 [그림 14-16]처럼 볼 수 있다.

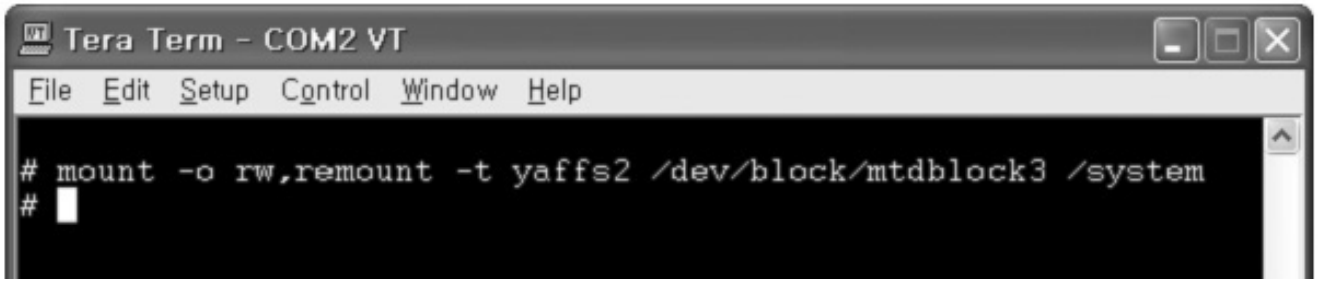
[그림 14-16] JNI 라이브러리 생성

14.4 NDK 고급 실습

- **[실습 14-5] JNI 라이브러리 생성 및 다운로드**

  (5) mount -o rw,remount -t yaffs2 /dev/block/mtdblock3 /system

  위에서 생성한 libled-jni.so 라이브러리 파일을 사용하기 위해서는 라이브러리 파일을 H-AndroSV210에 옮겨서 실행하여야 한다. adb를 사용하여 H-AndroSV210에 연결하여 생성된 libled-jni.so 파일을 다운로드한다.

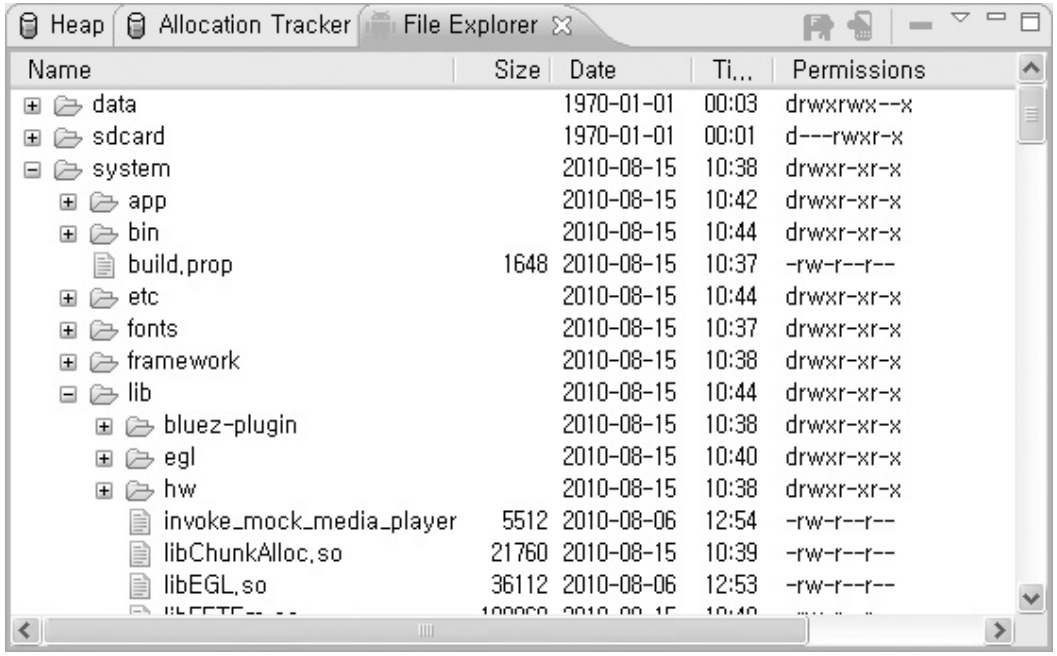


[그림 14-17] libled-jni.so 라이브러리 파일 타깃으로 이동하기

14.4 NDK 고급 실습

- **[실습 14-5] JNI 라이브러리 생성 및 다운로드**

  (6) 위의 작업을 진행하였으면 adb를 사용하여 위에서 생성한 libled-jni.so 파일을 /system/lib 이하로 [그림 14-18]처럼 이동한다.



[그림 14-18] libled-jni.so 파일 저장

14.4 NDK 고급 실습

- **[실습 14-6] LED 제어 안드로이드 애플리케이션 생성**

  (1) cd /mnt/cdrom/SV210_Android_100822/APP

  LED를 제어하기 위한 소스가 APP 폴더에 Source.tgz 파일로 압축되어 있다.

  (2) tar zxvf Source.tgz

  [그림 14-19]처럼 /SV210_Test_app라는 폴더 이하에 압축이 해제된다.

[그림 14-19] LED 제어 애플리케이션 파일 Source.tgz 압축 해제

14.4 NDK 고급 실습

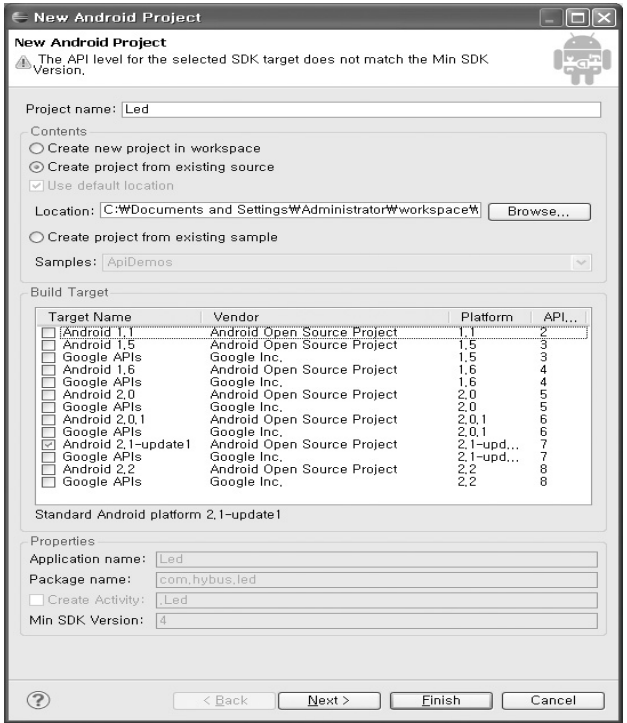
- **[실습 14-6] LED 제어 안드로이드 애플리케이션 생성**

  (3) cd SV210_Test_app; ls

  해당 폴더로 이동하여 폴더 내용을 확인한다.

  (4) Led 폴더를 이클립스의 프로젝트에 추가한다.

  이클립스 메뉴에서 File → New → Project → Android Project 화면에서 'Contents'항목의'Create project from existing source'를 선택하고, Location에서 추가할 프로젝트의 경로를 지정해준다. [그림 14-20]처럼 설정이 되어 있는지 확인한 후에'Finish'버튼을 누른다.



[그림 14-20] LED_Test 프로젝트 추가

14.4 NDK 고급 실습

- **[실습 14-6] LED 제어 안드로이드 애플리케이션 생성**

(5) 프로젝트가 정상적으로 추가되었으면 좌측의 Workspace 화면에 Led라는 프로젝트가 추가된 것을 볼 수 있다.

(6) Led.java를 확인한다.

추가된 Led라는 프로젝트 아래의 src/com.hybus.led/Led.java 파일을 오픈하면 다음과 같은 내용을 확인할 수 있다. 다음은 Led.java의 전체 소스의 내용이다.

```
package com.hybus.led;
import android.app.Activity;
import android.widget.CheckBox;
import android.widget.ToggleButton;
import android.os.Bundle;
import android.view.View;
public class Led extends Activity
{    /** Called when the activity is first created. */
 int LedData;
        @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        System.loadLibrary("led-jni");

        LedData = 0x70;
        LEDTest(LedData);

        final ToggleButton Red = (ToggleButton)findViewById(R.id.RedLED);
        final ToggleButton Yellow = (ToggleButton)findViewById(R.id.YellowLED);
        final ToggleButton Blue = (ToggleButton)findViewById(R.id.BlueLED);

        Red.setChecked(true);
        Red.setOnClickListener(new CheckBox.OnClickListener() {
                public void onClick(View v){
                        if(Red.isChecked())
                        {
                                LedData |= (0x10);

                        }else
                        {
                                LedData &= ~(0x10);
                        }
                        LEDTest(LedData);
                }
        });

        Yellow.setChecked(true);
```

14.4 NDK 고급 실습

- **[실습 14-6] LED 제어 안드로이드 애플리케이션 생성**

  (7) 실행

  애플리케이션은 라이브러리 파일에 접근하여 라이브러리 내에 있는 내용을 호출한다. NDK로 작성된 애플리케이션은 C 언어 자체가 VM에서 실행되는 Java 보다 실행 속도가 빠르기 때문에 속도 면에서 유리하다. 하지만 NDK를 따로 빌드해야 하고 프로그래밍이 Java로만 작성하는 것보다 까다로운 단점이 있다. 실행 결과는 [그림 14-21]과 같다.

[그림 14-21] Led.java 실행 결과

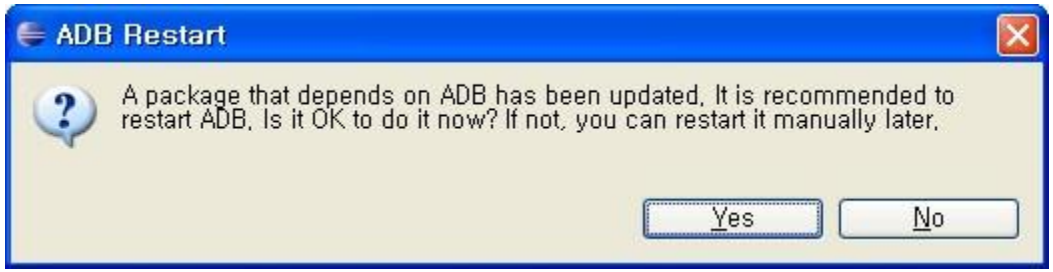
14.4 NDK 고급 실습

안드로이드 SDK 업데이트

- SDK 파일을 다운로드한 후 설치하는 과정이 다소 오랜 시간(20~40분) 소요(각종 레벨의 안드로이드 SDK 플랫폼, 문서, Google API를 다운로드 및 설치)

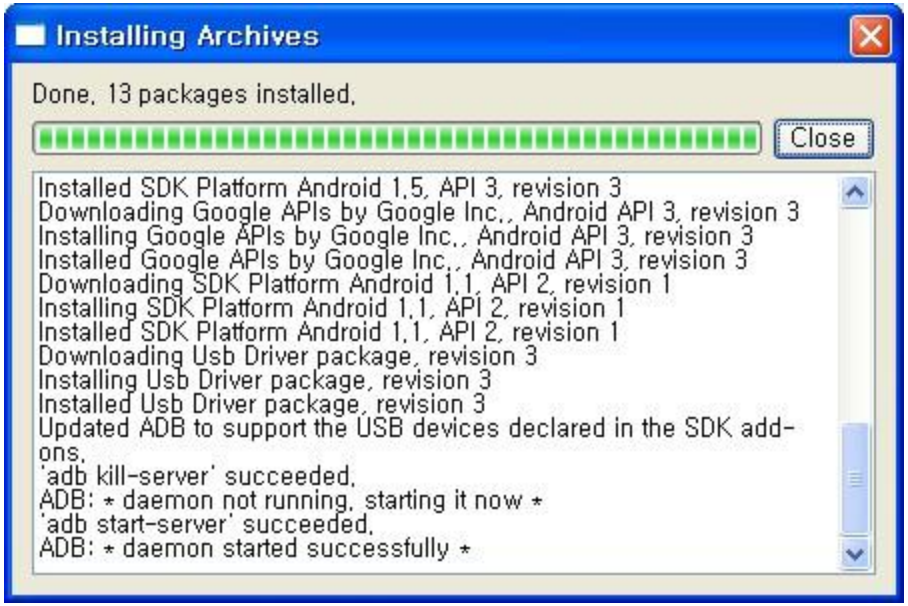
안드로이드 SDK 업데이트

- ADB(Android Debug Bridge) 재시작 창이 나타나면 [Yes] 클릭



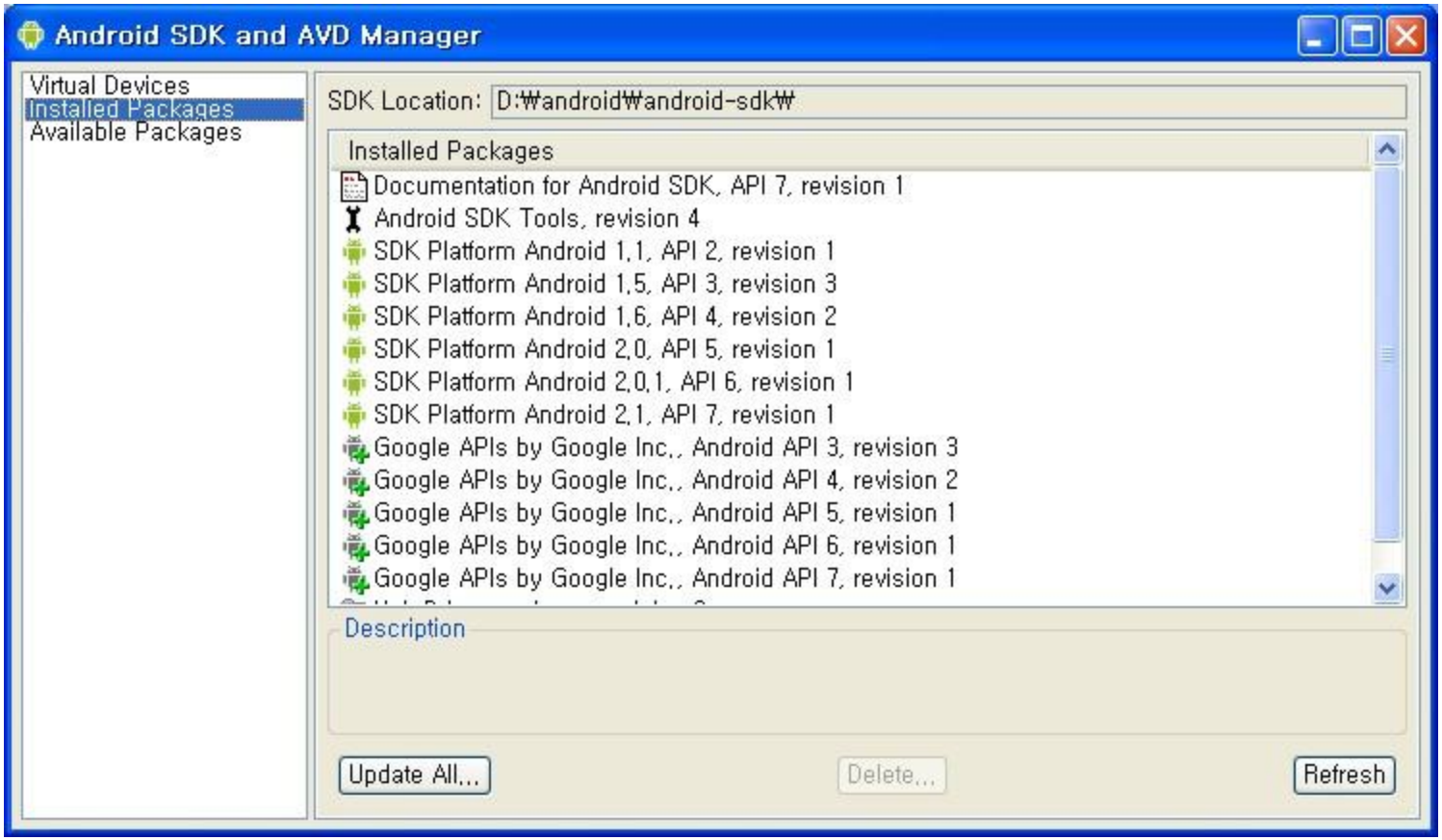
안드로이드 SDK 업데이트

- USB 장치를 지원하기 위한 ADB를 업데이트하면 Installing Archives 창을 닫는다



안드로이드 SDK 업데이트

- Android SDK and AVD Manager 창의 좌측에 있는 Installed Packages를 선택하면 업데이트된 SDK 목록이 나타남(14개)



AVD와 SD 카드 생성

AVD 생성

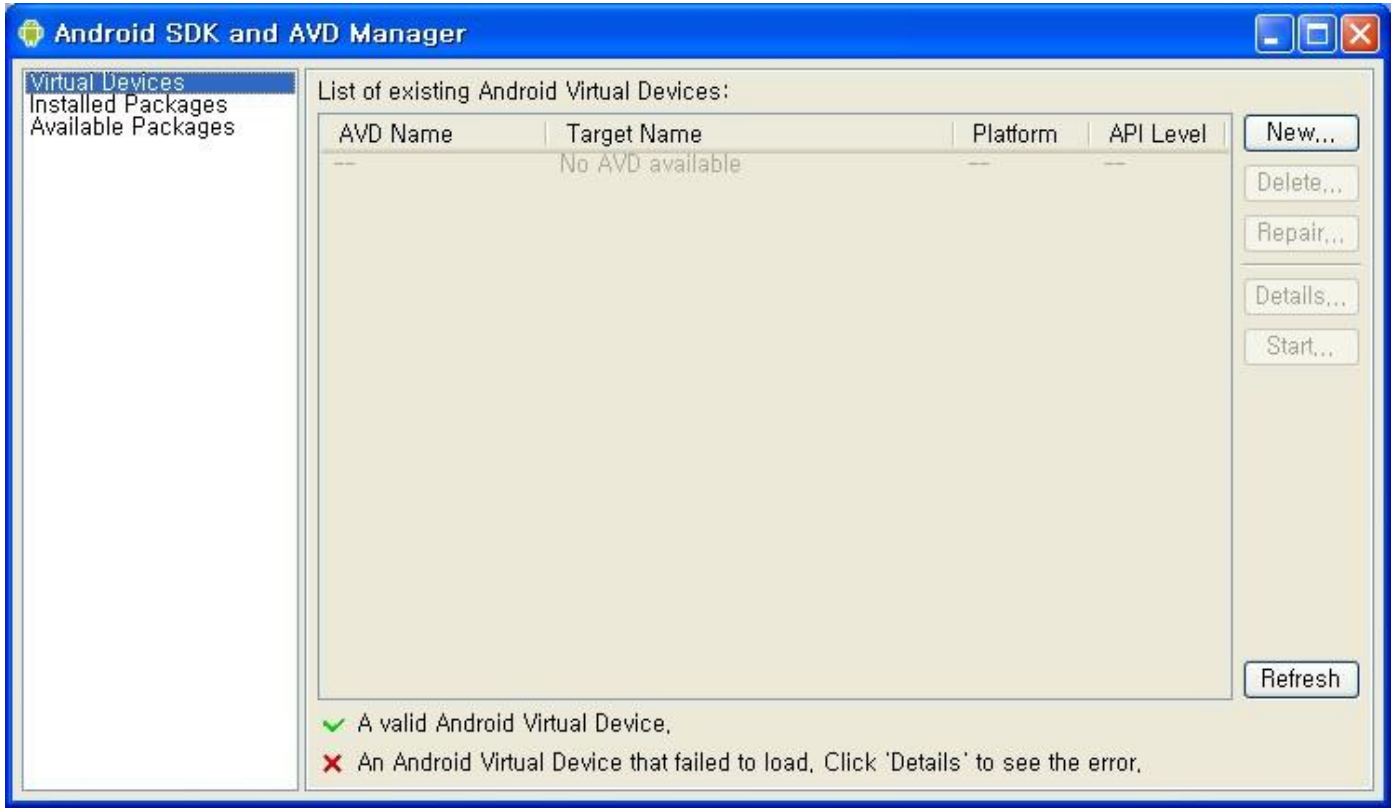
- **AVD란**
  - Android Virtual Devices
  - SDK 1.5부터 에뮬레이터를 실행하기 위하여 최소 1개의 AVD 필요
  - AVD로 인하여 여러 버전의 안드로이드 디바이스를 위한 애플리케이션을 각각의 버전과 SDK Add-On에 맞게 테스트 가능
  - 예를 들어 카메라가 있는 경우, 쿼티 자판이 있는 경우, 1.1 SDK 탑재한 단말, 1.5 SDK를 탑재한 디바이스 등 여러 가지 구성을 가지고 있는 가상의 디바이스를 지원 가능하게 함
  - 각 AVD마다 하나의 안드로이드 에뮬레이터를 구동할 수 있음

AVD 생성

- 이클립스 툴바에서 [Android AVD and SDK Manager] 버튼을 클릭

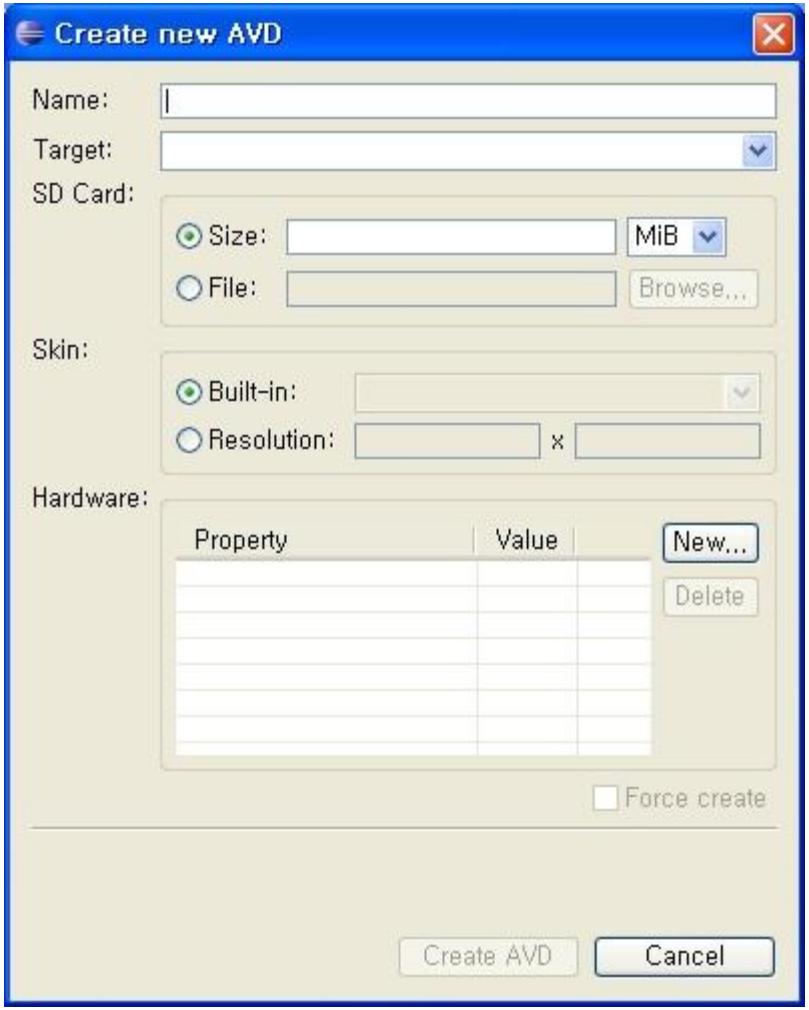
AVD 생성

- Android AVD and SDK Manager 창의 우측에 있는 [New...] 버튼을 클릭



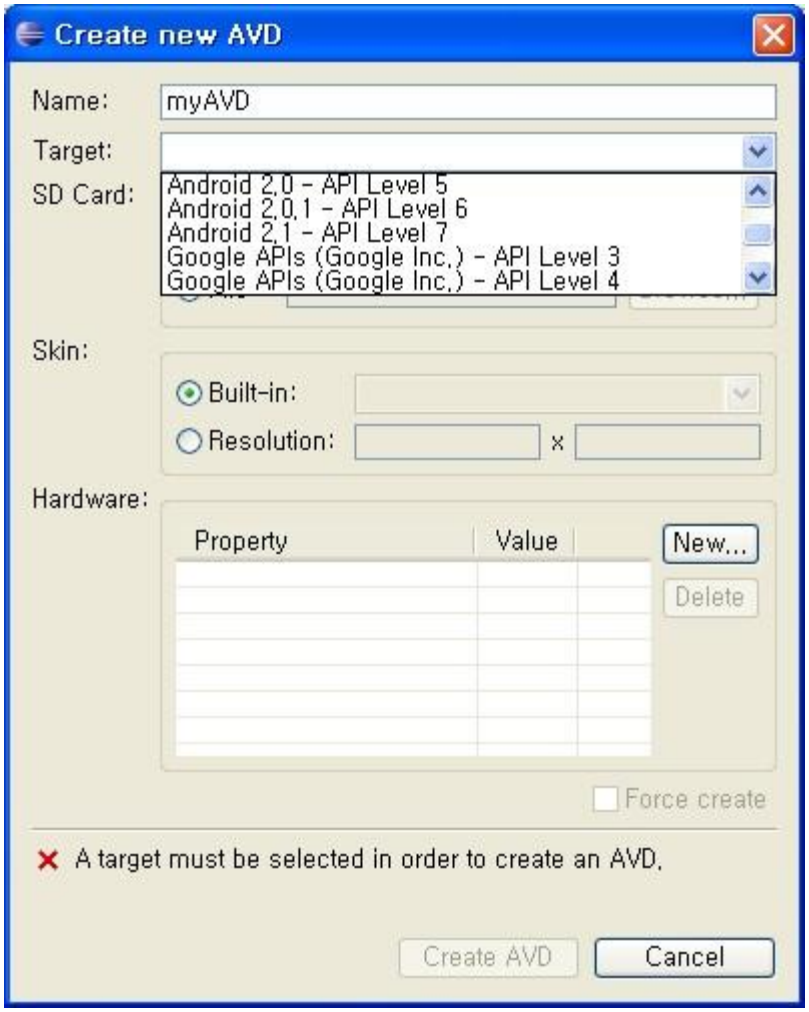
AVD 생성

- Create new AVD 창의 Name 항목에 적절한 디바이스 이름 선택



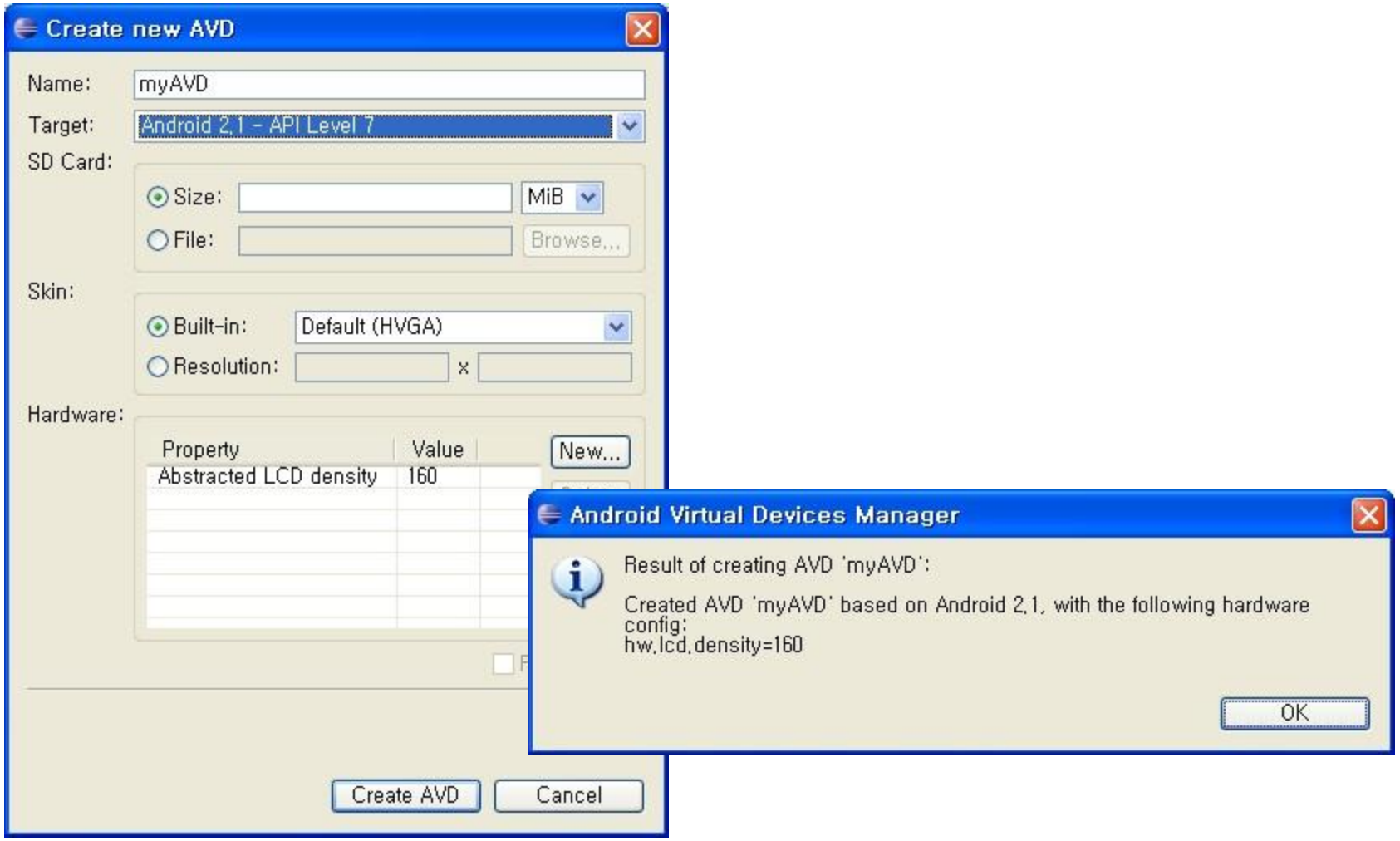
AVD 생성

- Target에 사용할 SDK 플랫폼 API 버전 혹은 Google API 버전을 선택



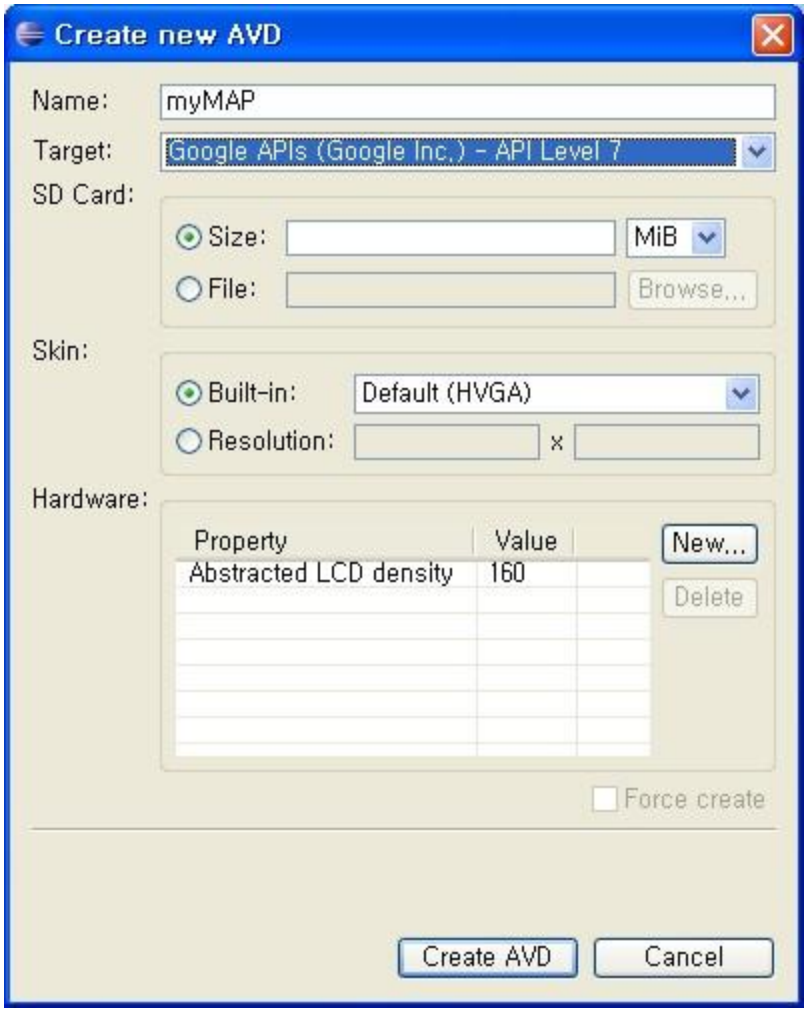
AVD 생성

- AVD 생성 완료



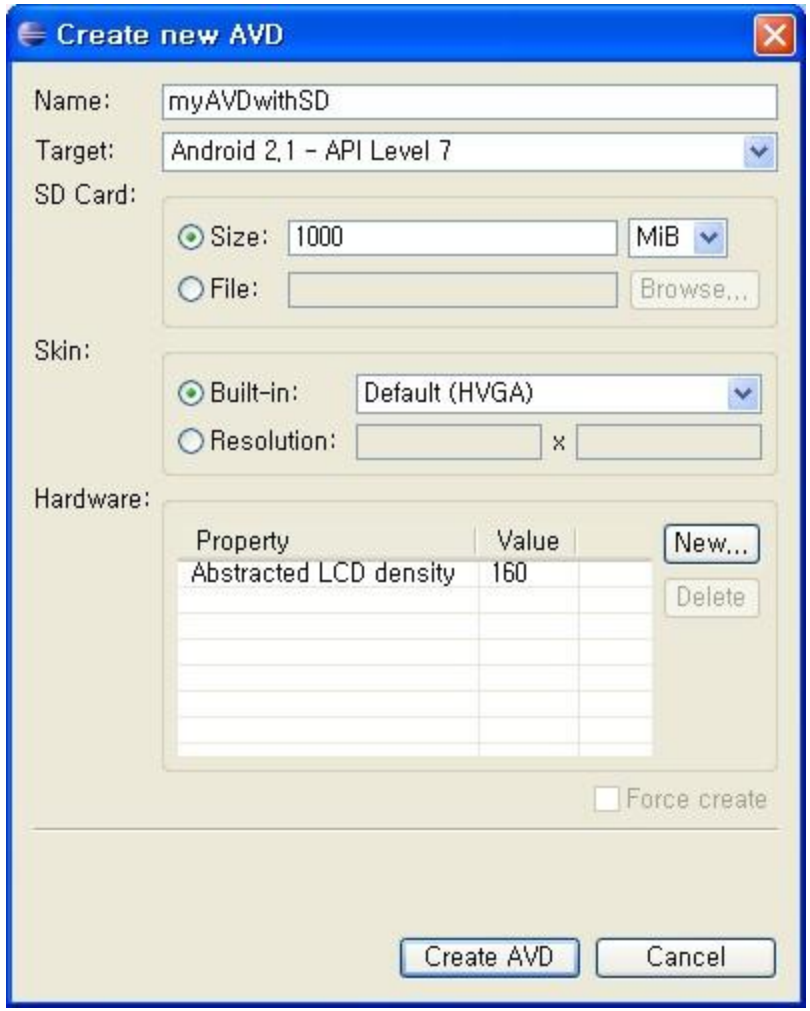
AVD 생성

- Google Map API를 사용하기 위한 AVD 생성



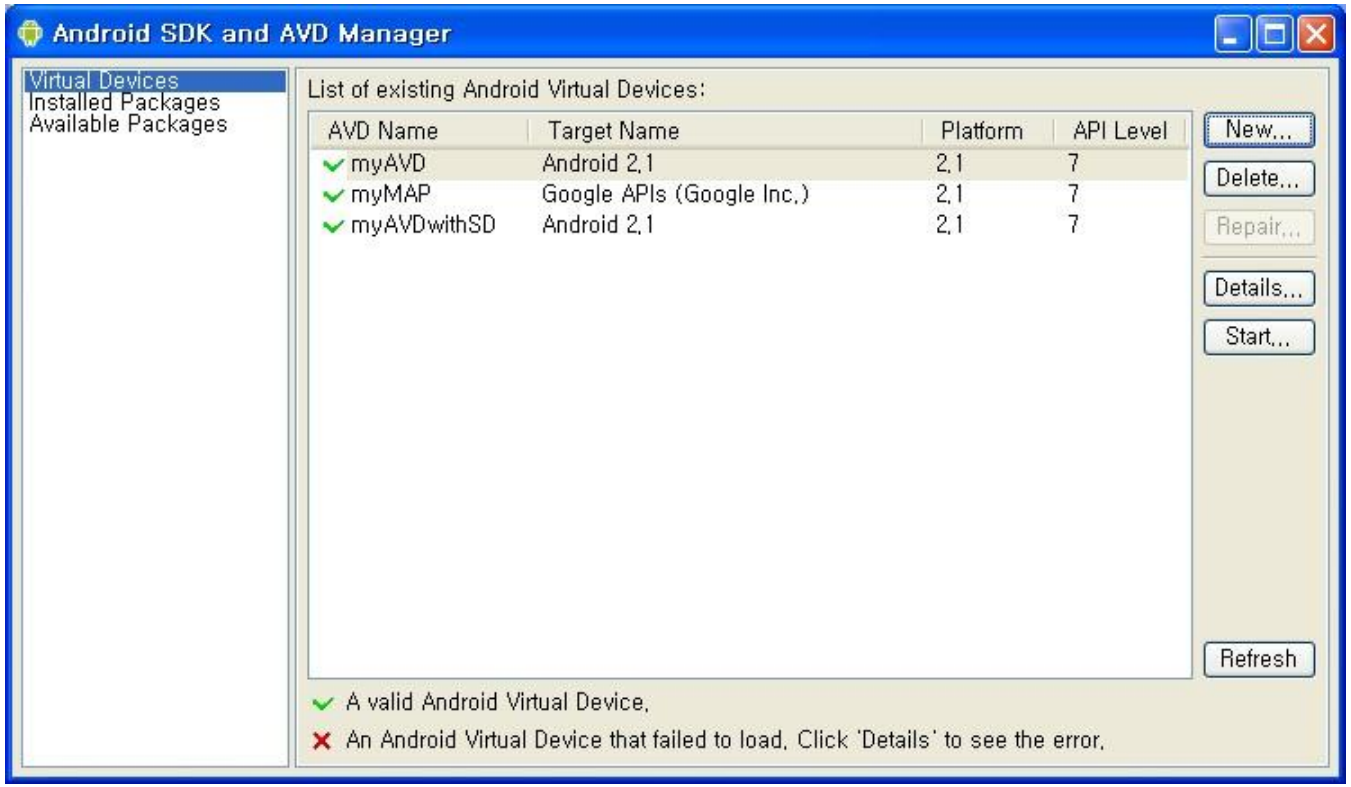
가상 SD 카드 생성

- **SD 카드의 생성**
  - SD 카드의 크기를 입력
  - 혹은 기존에 생성한 SD 카드를 사용하려면 File을 선택한 후 [Browse...] 버튼을 눌러 가상 SD 카드 파일을 선택



AVD와 가상 SD 카드 확인

- 이클립스 툴바에서 [Android AVD and SDK Manager] 버튼을 클릭 → 왼쪽의 Virtual Devices 선택



안드로이드 SDK 둘러보기

안드로이드 SDK란?

- 안드로이드 애플리케이션의 개발, 테스트, 디버그 하는데 필요한 모든 API들과 도구를 포함
- 안드로이드 SDK 플러그인을 사용하여 Eclipse IDE에 적용 가능
- 안드로이드 SDK 폴더

안드로이드 SDK란?

- **주요 구성 요소**
  - 안드로이드 API
    - 구글이 네이티브 안드로이드 애플리케이션을 개발하기 위하여 사용한 것과 동일한 라이브러리
  - 개발도구
    - 안드로이드 응용 프로그램 개발, 즉 애플리케이션의 컴파일, 디버그, 테스트하는데 필요한 각종 도구
    - 이클립스상에서 개발하는 경우 이 도구들을 직접 다루지는 않음
  - 풍부한 문서
    - 안드로이드 SDK의 각종 패키지, 클래스에 대한 설명
    - 안드로이드 개발을 시작하는 방법과 원리를 설명
    - Java의 경우 Java SDK Documentation과 유사
  - 샘플 코드
    - 안드로이드 API 기능의 사용법을 나타낸 예제 프로그램 코드
    - 이클립스 프로젝트에 추가하여 실행 가능

주요 안드로이드 도구

- **emulator.exe**
  - Dalvik 가상머신의 구현으로 하드웨어 중립적
  - 안드로이드용 응용 프로그램이 실제 휴대단말에서 동작하는 모습을 PC로 확인
  - 옵션을 포함하여 실행 가능하지만 대부분 이클립스에서 구동
  - 구동하는 것은 리눅스 시스템을 부팅하는 것이기 때문에 장시간 요구. 에뮬레이터를 구동한 후 계속 작업 가능

주요 안드로이드 도구

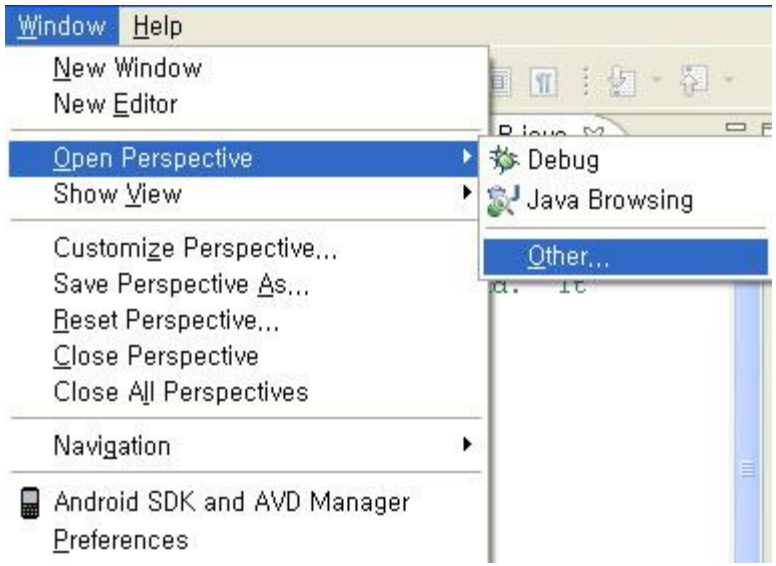
- **adb.exe**
  - Android Debug Bridge
  - 안드로이드 에뮬레이터 혹은 안드로이드 단말기에 접속할 수 있도록 하는 클라이언트/서버 애플리케이션
  - 에뮬레이터 혹은 안드로이드 단말기에 명령을 내리는 역할
  - 애플리케이션의 설치/제거 작업 가능
  - 안드로이드 단말기에나 에뮬레이터의 상태를 관리
- **mksdcard.exe**
  - 하드디스크의 일부분을 안드로이드 에뮬레이터에서 가상의 SD 카드로 생성
- **dx.bat**
  - Dalvik VM에 구동할 수 있는 응용 프로그램으로 만들어주는 컴파일러
  - 컴파일이 완료되면 *.dex의 확장자를 가진 파일을 생성

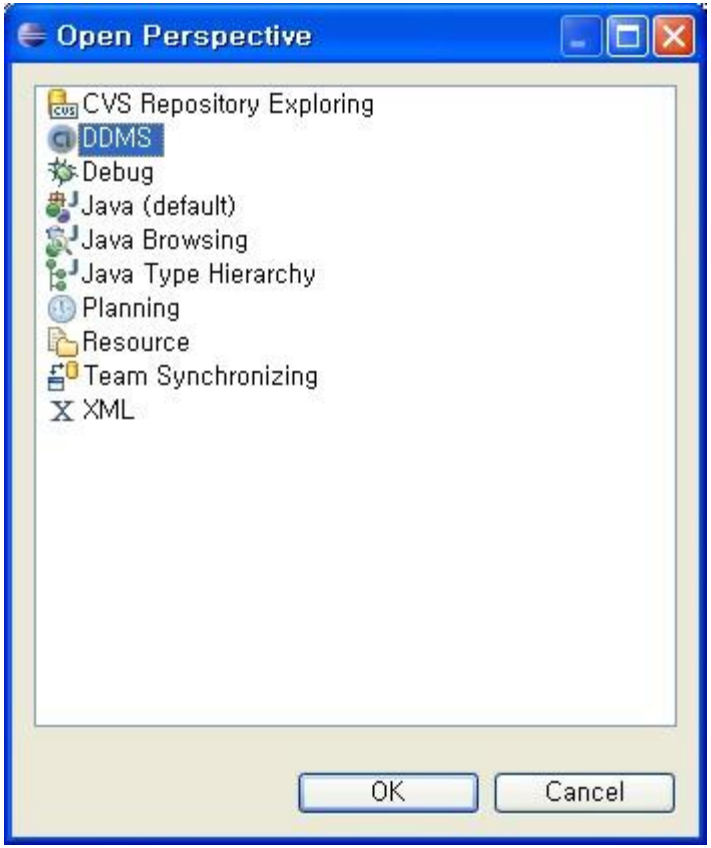
주요 안드로이드 도구

- **aapt.exe**
  - Android Asset Packaging Tool
  - 배포 가능한 안드로이드 패키지 파일(*.apk)을 생성
- **aidl.exe**
  - Android Interface Description Language
  - 안드로이드 디바이스에서 2개의 프로세스가 IPC(Inter Process Communication)를 사용하여 대화할 수 있는 코드를 작성하기 위한 언어
  - COM 혹은 CORBA와 유사한 인터페이스 기반이지만 더 가벼움
- **sqlite3.exe**
  - SQLite 데이터베이스 파일을 제어하는 도구

주요 안드로이드 도구

- **ddms.bat**
  - Dalvik Debug Monitor Service
  - 활성화된 스레드를 감시 및 중단하는 도구
  - 활성화된 모든 에뮬레이터의 파일시스템을 탐색하는 도구
  - DDMS 사용





주요 안드로이드 도구

- **ddms.bat**
  - DDMS 화면