

# LẬP TRÌNH ANDROID NDK

Giảng viên: Quách Công Hoàng

Trường Đại học Công nghệ - ĐHQG Hà Nội

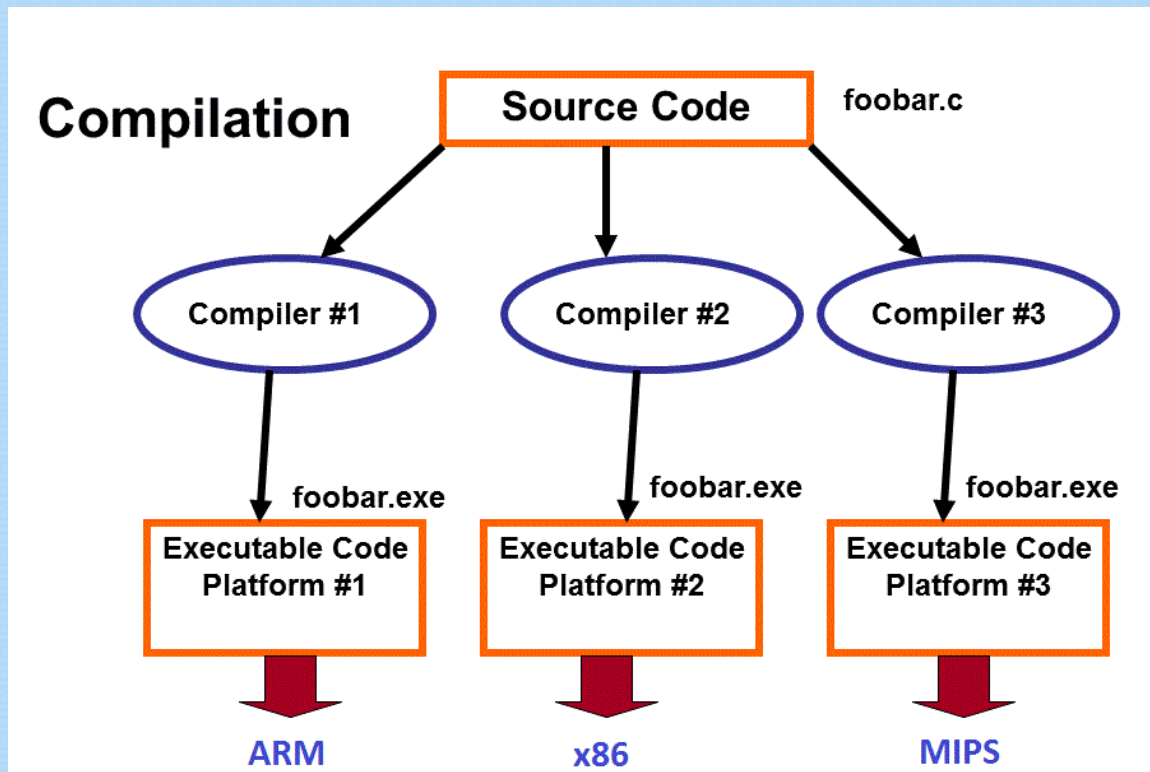
# Native Android Application

# 1. Tổng quan về NDK

- **Android NDK:** là một bộ công cụ cho phép thực thi một số phần trong ứng dụng Android bằng native-code (được viết chủ yếu bằng ngôn ngữ C/C++) thay vì byte-code (viết bằng Java).
- **Native-code:** là mã lập trình được biên dịch để chạy trên một kiến trúc CPU cụ thể (MIPS, ARM, x86)
- **Byte-code:** trong ứng dụng Android, mã nguồn được biên dịch thành mã máy ảo Java (Java bytecode). Máy ảo Java chịu trách nhiệm chuyển bytecode sang mã máy tương ứng (native-code).
  - Google phát triển các máy ảo Java (Dalvik, ART) chạy chương trình trên các kiến trúc CPU khác nhau.

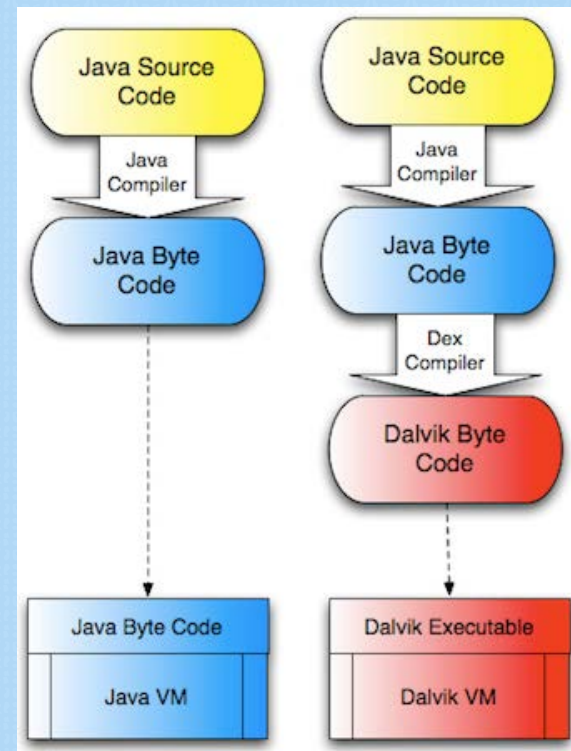
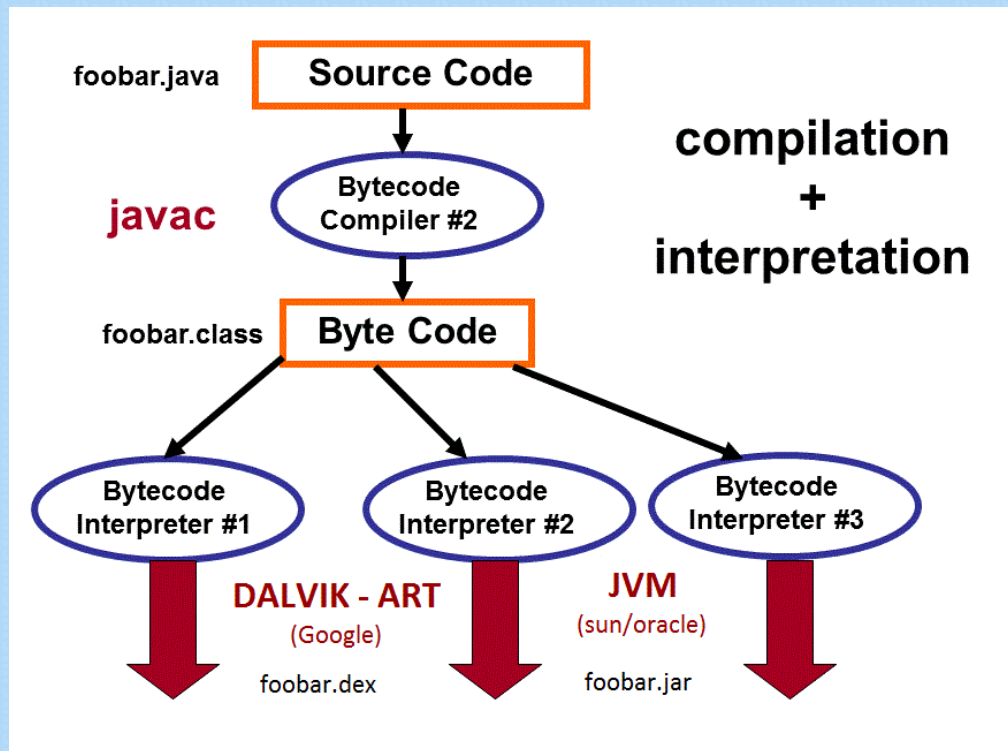
# 1. Tổng quan về NDK (tiếp)

- Biên dịch C/C++



# 1. Tổng quan về NDK (tiếp)

- Biên dịch Java



# 1. Tổng quan về NDK (tiếp)

- Việc sử dụng NDK để sử dụng code C/C++ trong ứng dụng Android có thể giúp ích đáng kể cho lập trình viên trong một số trường hợp sau:
  - Chuyển đổi ứng dụng sang nhiều nền tảng khác nhau.
  - Sử dụng các thư viện được cung cấp bởi các lập trình viên khác (hoặc cung cấp thư viện cho các lập trình viên khác sử dụng lại)
  - Tăng cường hiệu năng trong một số trường hợp nhất định, đòi hỏi CPU phải hoạt động tối đa, thí dụ như: games, xử lý tín hiệu số ...

## 2. Các phần chính của NDK

1. **Ndk-build:** là công cụ chính của NDK, có nhiệm vụ chuyển đổi các script-build thành các câu lệnh biên dịch tương ứng. Các script này có nhiệm vụ:

- Kiểm tra các yêu cầu biên dịch của lập trình viên để xác định cụ thể các tùy chọn trong việc biên dịch. Thí dụ: *Môi trường phát triển là gì (arm-v7, arm-v8 hay x86) ? Cần phải biên dịch các file mã nguồn nào ? Cần liên kết tới các thư viện nào ?*
- Biên dịch mã nguồn thành các file / thư viện mã nhị phân.
- Lưu các file nhị phân này vào project Android cần sử dụng.

Các script của ndk-build được lập trình viên tùy chỉnh trong file mặc định là *Android.mk* và *Application.mk*.

Android Studio tích hợp plugin cho NDK: *tùy chỉnh script-build trực tiếp bằng Gradle-script và không cần sử dụng file .mk nữa.*

2. **Native shared libraries:** là file thư viện với đuôi “.so” , được ndk-build biên dịch từ code C/C++ (tương đương với file “.dll” trên windows).
3. **Native static libraries:** là file thư viện với đuôi “.a” , được ndk-build biên dịch từ code C/C++ (tương đương với file “.lib” trên windows).
4. **Java Native Interface ("JNI"):** JNI là một giao tiếp cho phép Java và C/C++ có thể nói chuyện với nhau.
5. **Application Binary Interface ("ABI"):** ABI định nghĩa chính xác cách thức giao tiếp của mã máy với hệ điều hành và các thư viện hệ thống. Các ABI được phân biệt dựa vào đặc điểm của kiến trúc CPU: ARM, MIPS, x86.



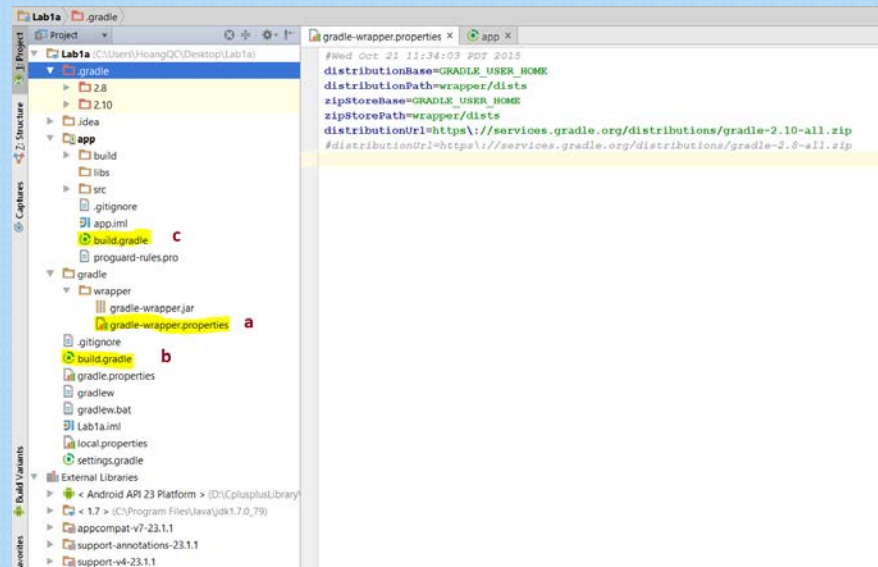
# 3. Native Android Project

**Project 1a:** Thiết lập Native Project với NDK và JNI trên IDE Android Studio.

- Bước 1: Khởi tạo project trên IDE.

Ghi chú:

- a) File tùy chọn cho Gradle sử dụng trong project.*
- b) File gradle-script cho toàn bộ project*
- c) File gradle-script cho module ứng dụng Android*



- Bước 2: nâng cấp Gradle cho project

- Vào (a) `./gradle/wrapper/gradle-wrapper.properties` thay đổi đường dẫn tới Gradle 2.9

```
distributionUrl=https\://services.gradle.org/distributions/gradle-2.9-all.zip
```

- Build | Make project (Ctrl + F9)

- Bước 3: thay đổi Gradle plugin mới cho project:

- Vào (b) `./build.gradle` sửa thành

```
classpath 'com.android.tools.build:gradle:1.5.0'
```

```
classpath 'com.android.tools.build:gradle-experimental:0.6.0-alpha3'
```

- Nếu chưa làm được bước 2 có thể tạm thời sửa thành:

```
classpath 'com.android.tools.build:gradle-experimental:0.4.0'
```

- Bước 3 (tiếp):

Thay đổi (c) *./app/build.gradle* cho phù hợp với phiên bản mới

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 23
    buildToolsVersion "23.0.2"

    defaultConfig {
        applicationId "com.example.uet.lab1a"
        minSdkVersion 22
        targetSdkVersion 23
        versionCode 1
        versionName "1.0"
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles
                getDefaultProguardFile('proguard-android.txt'),
                'proguard-rules.pro'
        }
    }
}
```



```
apply plugin: 'com.android.model.application'

model {
    android {
        compileSdkVersion = 23
        buildToolsVersion = "23.0.2"

        defaultConfig.with {
            applicationId = "com.example.uet.lab1a"
            minSdkVersion.apiLevel = 22
            targetSdkVersion.apiLevel = 23
            versionCode = 1
            versionName = "1.0"
        }
    }

    android.buildTypes {
        release {
            minifyEnabled = false
            proguardFiles.add(file('proguard-rules.txt'))
        }
    }
}
```

- Bước 4: Tạo Native Module trong gradle:

```
apply plugin: 'com.android.model.application'


model {
    android {
        compileSdkVersion = 23
        buildToolsVersion = "23.0.2"

        defaultConfig.with {
            applicationId = "com.example.uet.lab1a"
            minSdkVersion.apiLevel = 22
            targetSdkVersion.apiLevel = 23
            versionCode = 1
            versionName = "1.0"
        }
    }
}

android.ndk{
    moduleName = "myNativeLib"
    stl = "stlport_static"
}

android.buildTypes {
    release {
        minifyEnabled = false
        proguardFiles.add(file('proguard-rules.txt'))
    }
}
}
```

- Bước 5:
  - Khai báo hàm Native trong class Java (chú ý: khi thay đổi hàm Native cần phải biên dịch lại chương trình )



```
package com.example.uet.lab1a;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

    static {
        System.loadLibrary("myNativeLib");
    }
    public native String getStringFromNative();
    public native int getNumberFromNative();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

- Bước 5 (tiếp): dùng JNI để tạo file giao diện Java với C/C++
- Từ cửa sổ terminal truy cập vào thư mục ./app/src/main
- Gõ command-line của JNI tới class Java được tạo hàm Native

```
javah -d jni -classpath %ANDROID_HOME%\platforms\android-23\android.jar;%ANDROID_HOME%\extras\android\support\v7\appcompat\libs\android-support-v7-appcompat.jar;%ANDROID_HOME%\extras\android\support\v4\android-support-v4.jar;..\..\build\intermediates\classes\debug com.example.uet.lab1a.MainActivity
```

- Cấu trúc của command:

**javah -d jni -classpath [...]** [class java có hàm native]

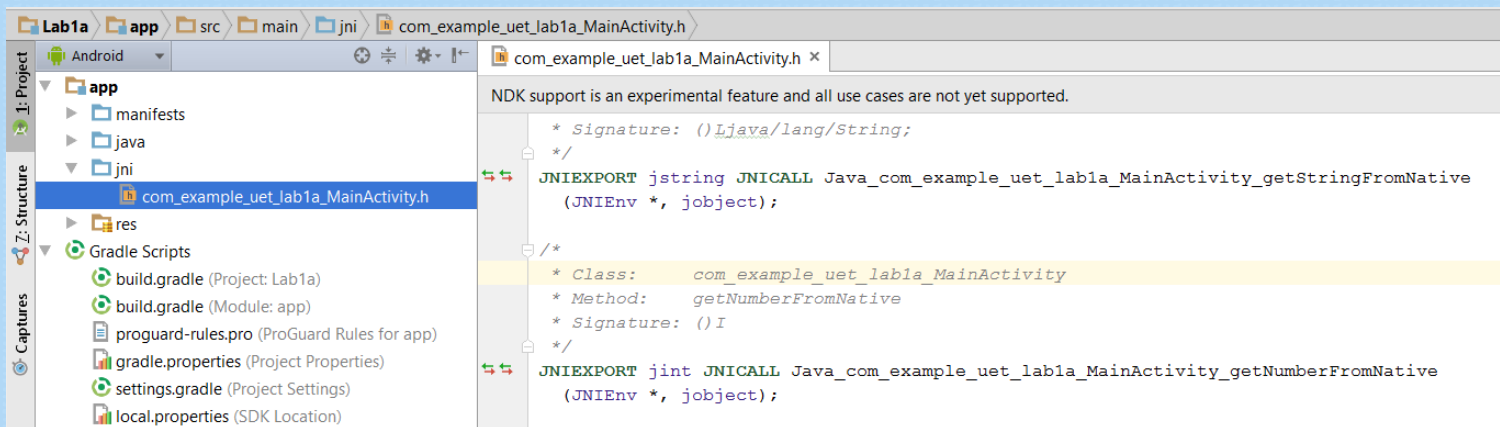
**-d jni** : tạo thư mục (directory) tên là jni

**-classpath** : đường dẫn tới thư mục chứa các class của app

*Chú ý: đường dẫn có thể thay đổi tùy thuộc vào:*

- + Đường dẫn hiện tại của SDK trên máy tính
- + Cấu hình gradle của project đang lập trình ( có thể biên dịch ra các file .class ở thư mục khác với thí dụ )

- Command javah sẽ tự động tạo thư mục JNI và file header trong đó có tên tương ứng class Java chứa hàm native:



- File header này chứa khai báo của các hàm native tương ứng với các hàm Native đã khai báo trong Java.

- Bước 6: viết code thực thi cho các hàm Native
  - Thêm file \*.cpp vào thư mục ./app/src/main/jni
  - Nội dung file \*.cpp thêm vào có dạng như sau

```
#include "com_example_uet_labla_MainActivity.h"
#include "string"
#include "vector"

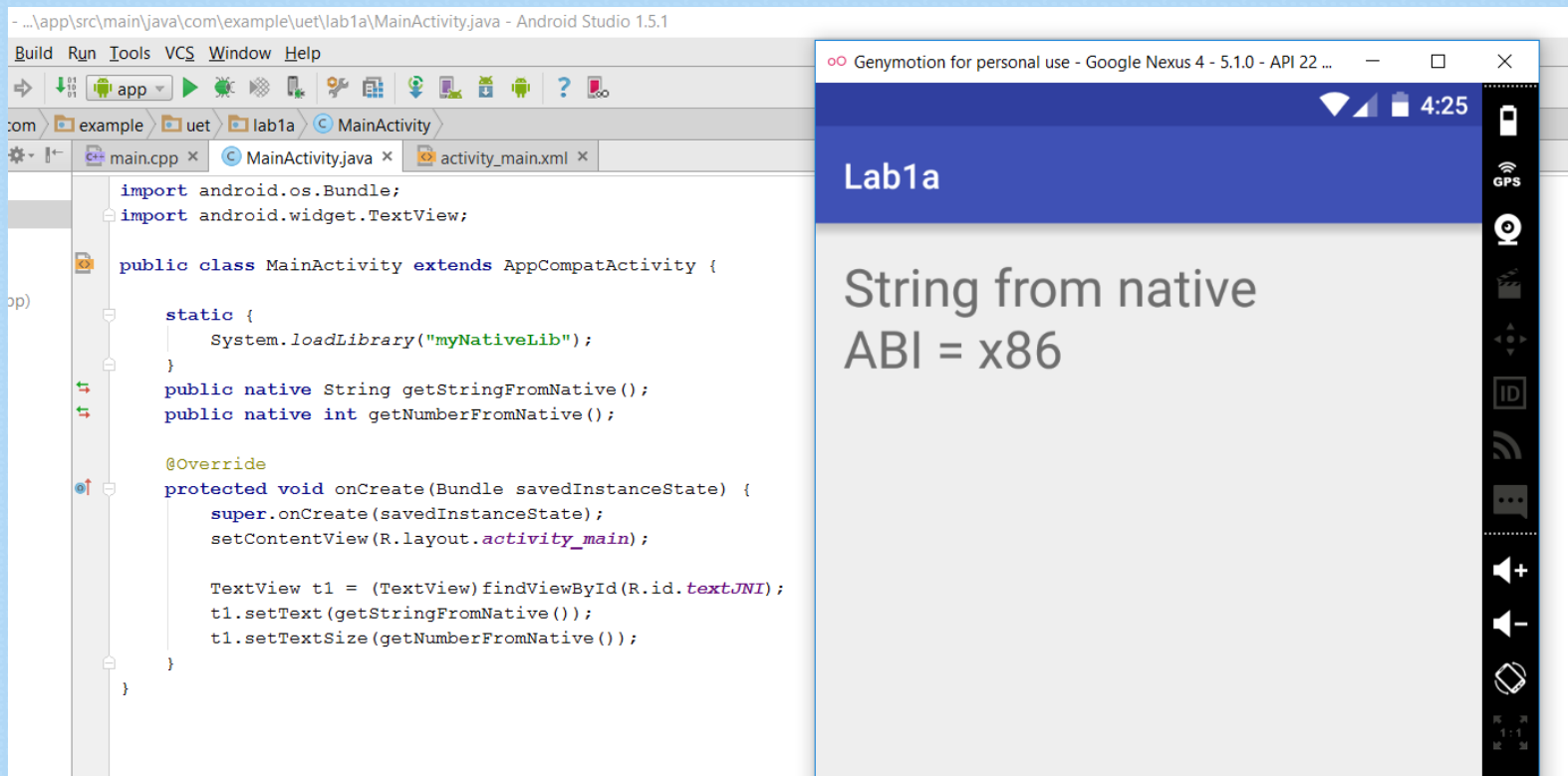
JNIEXPORT jstring JNICALL Java_com_example_uet_labla_MainActivity_getStringFromNative
(JNIEnv *env, jobject obj)
{
    std::string mString = "String from native";
    return (*env).NewStringUTF(mString.c_str());
}

JNIEXPORT jint JNICALL Java_com_example_uet_labla_MainActivity_getNumberFromNative
(JNIEnv *env, jobject obj)
{
    return 30;
}
```

- Lúc này các hàm Native đã được thực thi đầy đủ. Android app có thể sử dụng kết quả trả về từ các hàm này.



- Thí dụ:



## Project 1b:

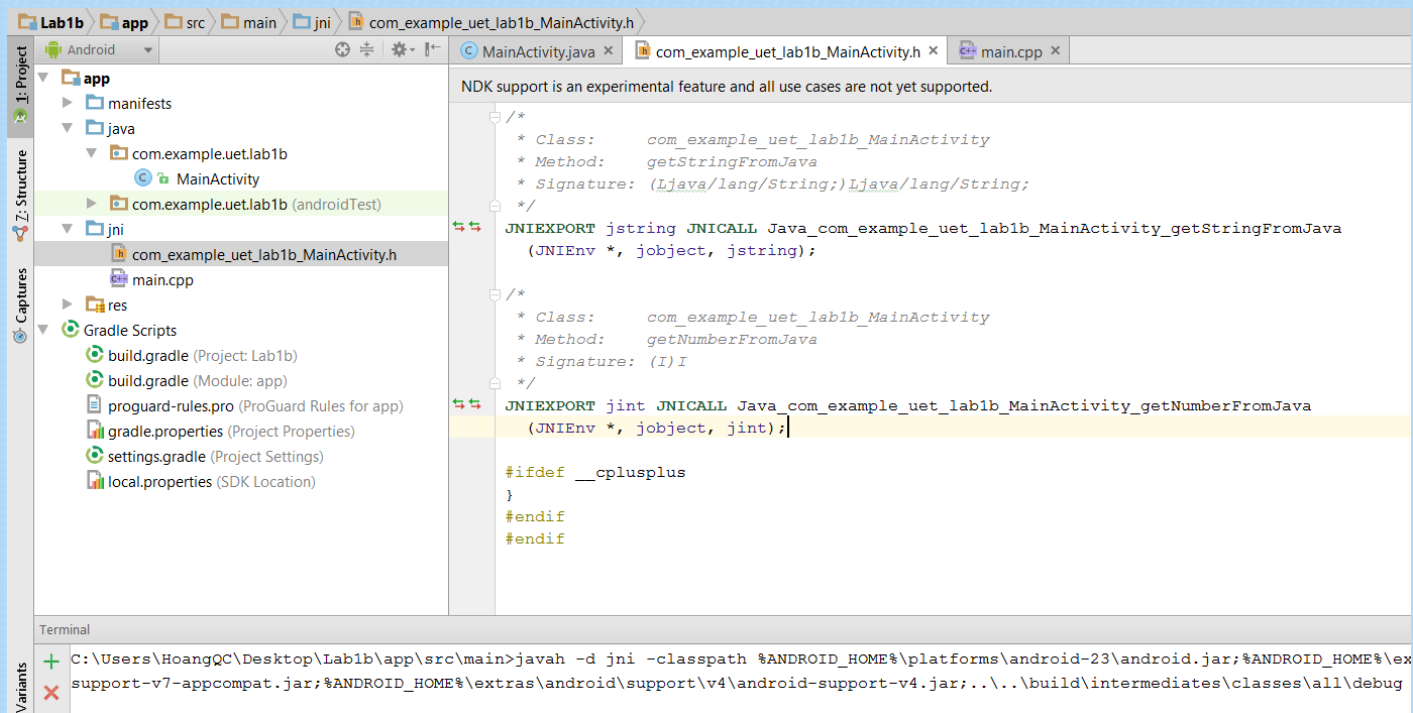
- Chuyển đổi các giá trị giữa Java và C/C++ thông qua JNI
- Sử dụng Android Studio để debug C/C++, thiết lập ABIs filter.

Triển khai project tương tự thí dụ trước. Sau đó thực hiện các bước sau:

- Bước 1: Khai báo thêm các hàm Native tham số đầu vào.

```
static {  
    System.loadLibrary("myNativeLib");  
}  
public native String getStringFromNative();  
public native int getNumberFromNative();  
  
public native String getStringFromJava(String inputString);  
public native int getNumberFromJava(int inputNumber);
```

- Bước 2: Biên dịch lại project (để cập nhật class Java)
- Bước 3: Dùng JNI command trên terminal để cập nhật lại file header cho class Java vừa biên dịch



- Bước 4: Viết code C/C++ cho hàm native vừa tạo

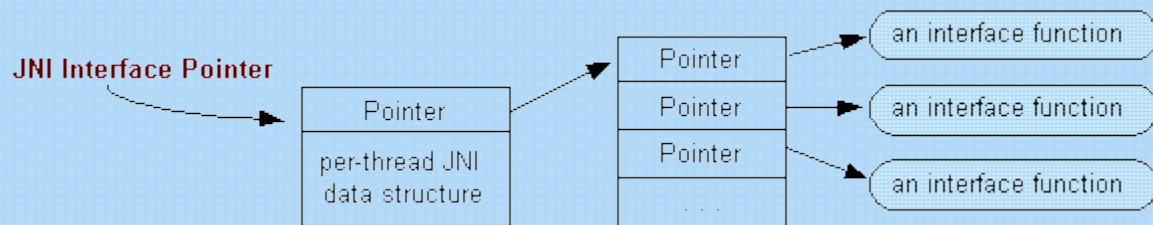
```
#include "com_example_uet_lab1b_MainActivity.h"
#include <string>

JNIEXPORT jstring JNICALL Java_com_example_uet_lab1b_MainActivity_getStringFromJava
(JNIEnv *env, jobject pThis, jstring jInputString)
{
    const char * inputChar = env->GetStringUTFChars(jInputString, NULL);

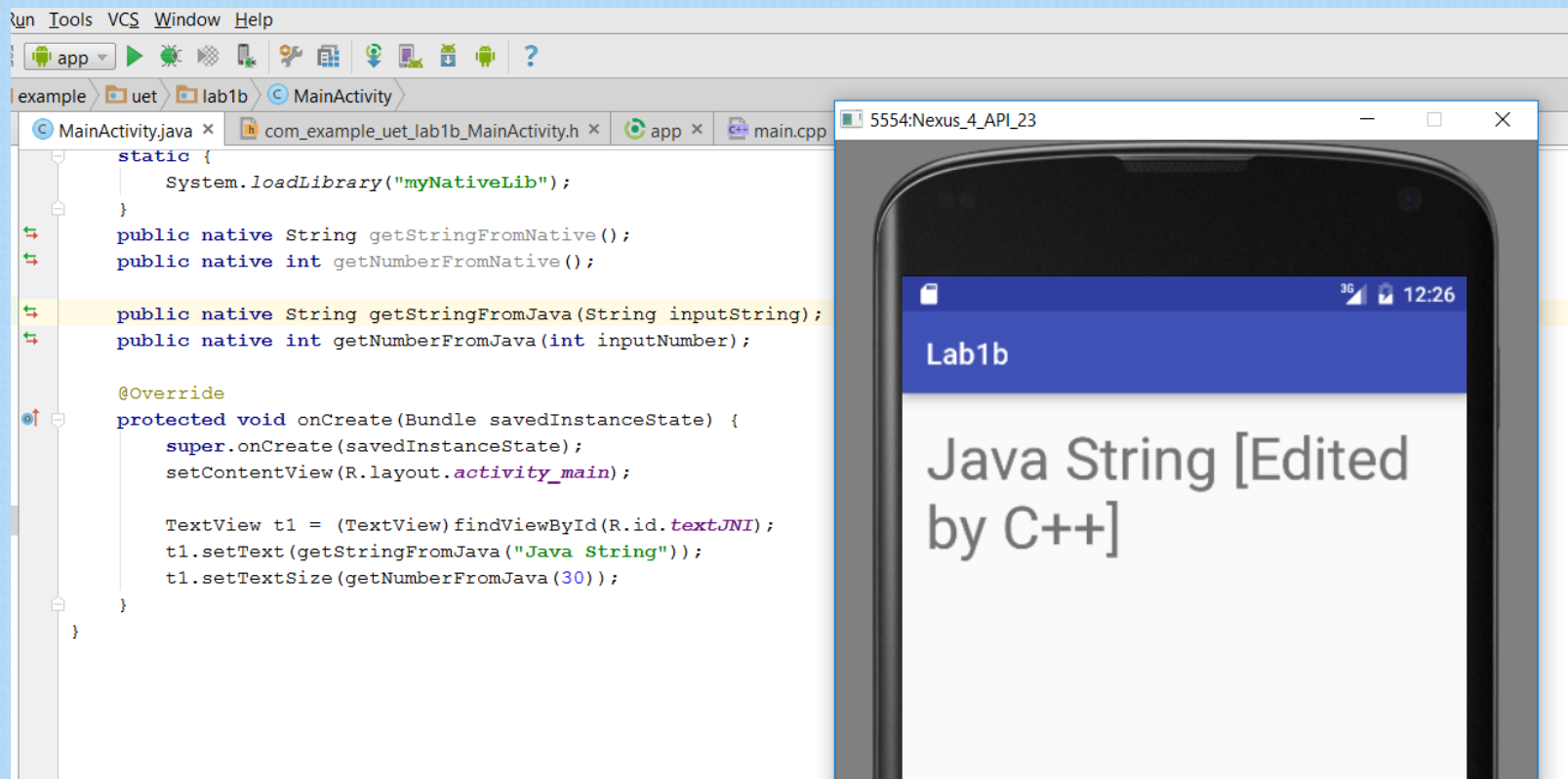
    std::string inputString(inputChar);
    std::string outputString = inputString + " [Edited]";
    const char * outputChar = outputString.c_str();

    env->ReleaseStringUTFChars(jInputString, inputChar);
    return (*env).NewStringUTF(outputChar);
}
```

*C/C++ tương tác với các đối tượng của Java (thí dụ như String) cần thông qua con trỏ JNIEnv để tham chiếu*



- Bước 5: Sử dụng hàm native trong Activity



- Bước 6.1 : Thiết lập chế độ Debug:
  - 6.1: Thay đổi gradle script như sau:

```
android.buildTypes {
    release {
        ndk.with{
            debuggable = true
        }
        minifyEnabled = false
        proguardFiles.add(file('proguard-
rules.txt'))
    }
}
```

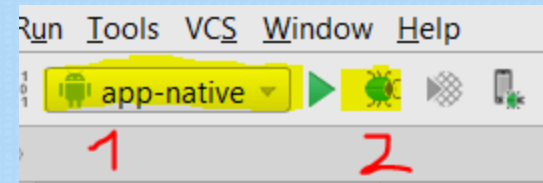
- 6.2 Đặt break-point tại vị trí cần theo dõi

```
JNIEXPORT jstring JNICALL Java_com_example_uet_lab1b_MainActivity
(JNIEnv *env, jobject pThis, jstring jInputString)
{
    const char * inputChar = env->GetStringUTFChars(jInputString,

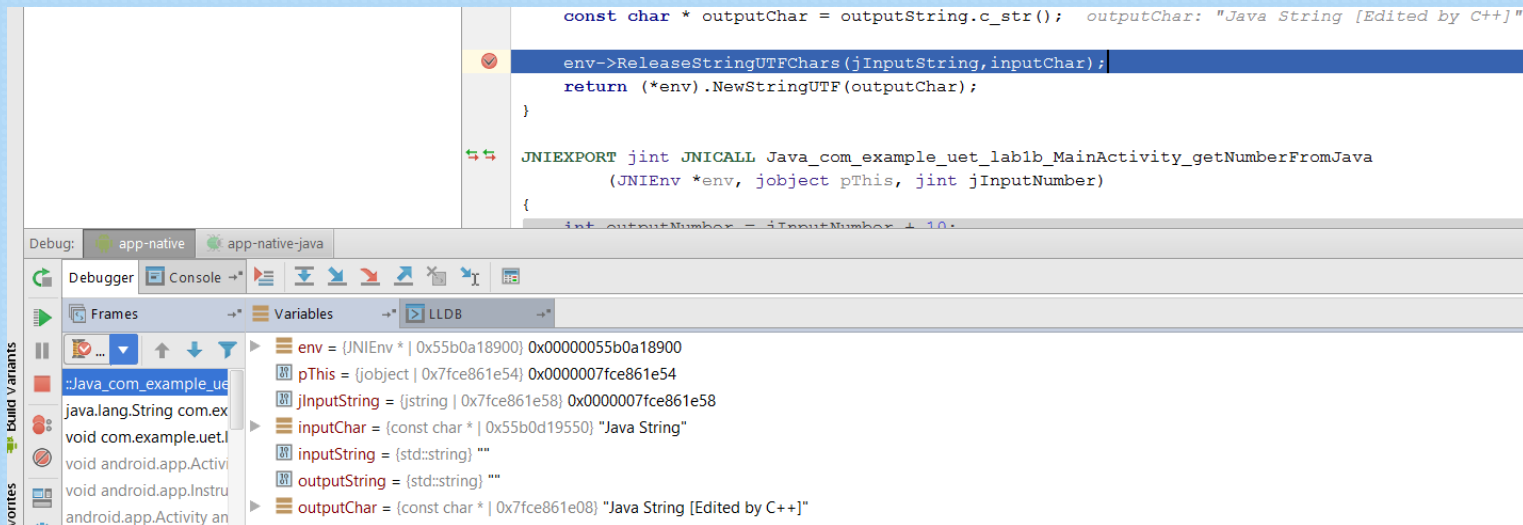
    std::string inputString(inputChar);
    std::string outputString = inputString + " [Edited by C++]";
    const char * outputChar = outputString.c_str();

    env->ReleaseStringUTFChars(jInputString, inputChar);
    return (*env).NewStringUTF(outputChar);
}
```

- 6.3: Chạy debug cho app-native



- Cửa sổ debug hiển thị các giá trị tại thời điểm Break-point



# Bài tập 1:

- Viết chương trình lưu danh sách sinh viên với yêu cầu sau:
  - Thông tin (họ tên, mã số SV) được nhập vào từ giao diện Java và lưu vào mảng C/C++
  - Có các API viết bằng JNI cho phép thêm bớt nội dung trong bản danh sách đã được lưu.



# Bài tập 2:

- Sử dụng thư viện TinyXML2 để viết chương trình đọc báo từ địa chỉ RSS sau: <http://vnexpress.net/rss/the-thao.rss>
- Giao diện và nội dung request URL xml được thực hiện bằng Java
- Việc phân tích nội dung (parsing) XML và lưu các kết quả phải được thực hiện bằng C/C++
  - Tài liệu về TinyXML2 tìm theo địa chỉ:  
<http://www.grinninglizard.com/tinyxml2/index.html>
  - Địa chỉ Github của TinyXML2:  
<https://github.com/leethomason/tinyxml2>