

Tips and Tricks for Building High-Performance Android Apps using Native Code

JD Hancock: Reign of The Android. <http://bit.ly/1GN8vmg>

Kenneth Geissshirt
kg@realm.io

Realm Inc.
[@realm](https://twitter.com/realm)
[http://github.com/Realm/](https://github.com/Realm/)
[http://realm.io/](https://realm.io/)

Today's Goal

- Android development is not just Java
- C and C++ can speed apps up
 - and reuse old legacy code
- Learn a few tricks when working with Java Native Interface
- Hands-on exercises



Swen-Peter Ekkebus: Goal! <http://bit.ly/1x1suYm>



@TechSection

Agenda



- Setting up your environment
- Java Native Interface

Exploratorium. After Dark: Time <http://bit.ly/1aSQxo0>

- Threading
- Using non-NDK libraries¹
- Benchmarking
- Debugging

¹NDK = Native Development Kit

Advantages

- Use C or C++ in your apps → no JIT needed as always machine instructions
- Access to system calls (like `mmap()`)
- Almost no limits on memory usage
- OpenGL programming
- Sharing code across platforms
- Reusing legacy libraries



Disadvantages



Rocher: That_Sux <http://bit.ly/1BdGNhJ>

- C and C++ memory management → easy to crash your app
- And debugging across languages is hard

- Bionic C library isn't the GNU C library
- Doing a native call is expensive (in time)
- Cumbersome to use non-NDK libraries

Use Cases

- AndEngine (2D Game Engine)

<http://www.andengine.org>

- ImageMagick for Android

<https://github.com/paulasiumwe/Android-ImageMagick>

- LevelDB for Android

<https://github.com/litl/android-leveldb>

- Support Vector Machine for Android

<https://github.com/cnbuff410/Libsvm-androidjni>



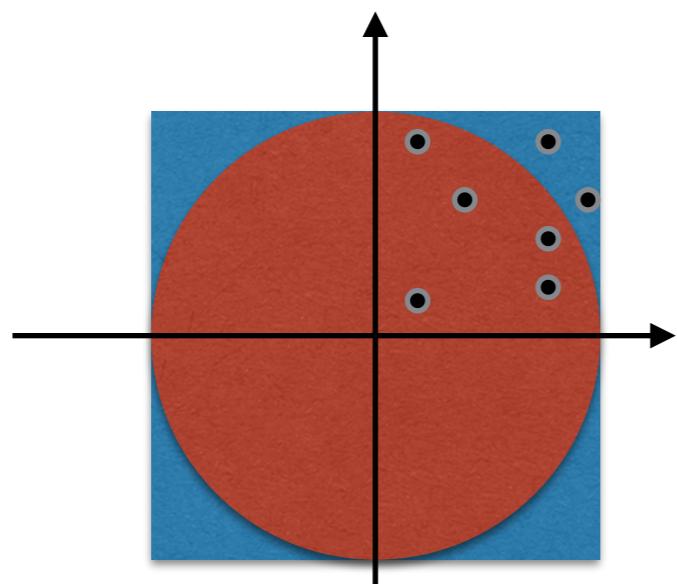
See-ming Lee; Rainbow Teddy shows BBF Stitch how to use the Canon EOS M; <http://bit.ly/1O3yA43>

Playing with C++

Example: estimating π

Estimating π using dart

- Throw arrows at random
- $N_{hit}/N = A/4$ (quarter of a circle)
- $A = \pi \cdot r^2$ (area of a circle)
- $\pi = 4 \cdot N_{hit}/N$ (with $r = 1$)



Bogdan Suditu: Darts. <http://bit.ly/18jCExj>

```
for i = 1, ..., N
    x = random(0, 1)
    y = random(0, 1)
    if (sqrt(x·x+y·y) < 1)
        Nhit = Nhit +1
π = 4 · Nhit/N
```

Example: the classes



jeremiah.andrick. My random number generator <http://bit.ly/1wbTbP2>

Random number generator

- Constructor with seed
- Method uniform () to return random number between 0.0 and 1.0

π estimator

- Constructor with number of iteration
- Method estimate () to calculate the value of π



Quinn Dombrowski: Pi <http://bit.ly/1Kz55cp>

Exercise 1

- Implement two C++ classes
 1. RNG - a random number generator
 - `uniform()` returns a number in $[0, 1[$
 2. pi - a π estimator using MC algorithm
 - `estimate()` returns π
- Write simple test program
- Compile using clang++ or g++
- Go to <https://github.com/kneth/AndroidCalcPi> if lazy 😊

The environment

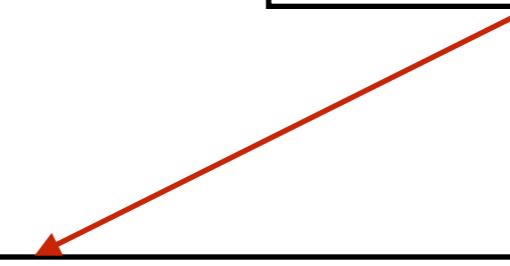
- Android Studio 1.3 Preview
- Java 1.7 JDK
 - or set version (► Build ► Build types)
- Android SDK 24.3.3
 - Android NDK can now be bundled
 - Gradle 2.5
 - A rooted device is nice to have
 - Always use Intel HAXM for the emulator



Dave Gingrich; <http://bit.ly/1HxRJWj>

```
ndk.dir=/usr/local/Cellar/android-sdk/24.3.3/ndk-bundle  
sdk.dir=/usr/local/Cellar/android-sdk/23.0.2
```

local.properties



Java Native Interface

Architecture

The Android App



MainActivity.java

Java wrapper classes



RNG.java

Pi.java

Intermediate C++ functions



net_zigzag_calcpi_NativeRNG.{h, cpp}

net_zigzag_calcpi_NativePi.{h, cpp}

Original C++ classes



rng.{hpp, cpp}

pi.{hpp, cpp}

Organize your files



- Use the experimental Android plugin for Gradle
- Place C/C++ files in app/src/main/jni/
- Add ndk.dir to local.properties
- Add an ndk section to your app's build.gradle

```
android.ndk {  
    moduleName = "calcPi"  
    cppFlags += ["-fexceptions"]  
    stl = "gnustl_shared" // for exceptions  
}
```

Building and Loading your native code

- Automatically build for all supported architectures (ARM, ARMv7, ARM64, MIPS, MIPS64, x86, x86-64)
- Android only installs the relevant architecture
- Load native code early (main activity's onCreate)



Thomas Hawk: Loading Zone <http://bit.ly/1MoBk9S>

```
System.loadLibrary("calcPi");
```

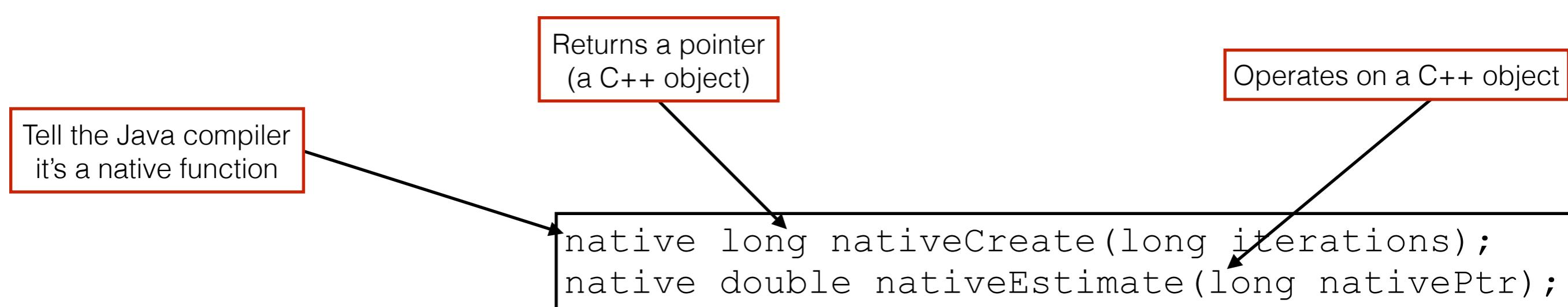
The module name (from build.gradle)

Calling a C++ function

- Use Java's native keyword in the signature
- Implement the function in C++
- Use a long to store the C++ pointer when you creates an object



lamont_cranston: call center <http://bit.ly/1A4JIK5>



Generate header files

- .class files contain information about native calls
- javah can extract signatures and generate C/C++ header files



Paul: Generator Room Door <http://bit.ly/1C3SSs7>

Add path to jar files you depend on

```
#!/bin/bash
# Setting up
CLASSDIR="$ (pwd) /../app/build/intermediates/classes/debug"
JNIDIR="$ (pwd) /src"

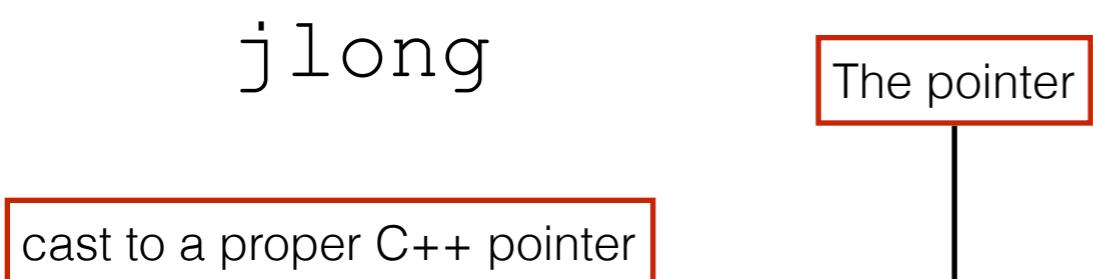
# Generate the headers
(cd "$CLASSDIR" && javah -jni -classpath "$CLASSDIR" -d "$JNIDIR"
net.zigzak.calcpi.NativePi net.zigzak.calcpi.NativeRNG)

# Remove "empty" header files (they have 13 lines)
wc -l "$JNIDIR"/*.* | grep " 13 " | awk '{print $2}' | xargs rm -f
```

Classes to extract from

Java types in C/C++

- Java types are mapped to C/C++ types
- Convert Date to long before call
- Pass pointers over as long/jlong



Java	C/C++
long	jlong
String	jstring
long[]	jlongArray
Object	jobject
float	jfloat

```
JNIEXPORT jdouble JNICALL Java_net_zigzag_calcpi_NativePi_nativeEstimate  
    (JNIEnv *, jobject, jlong nativePtr)
```

```
{  
    Pi *pi = reinterpret_cast<Pi *>(nativePtr);  
    double estimate = pi->estimate();  
    return estimate;  
}
```

Define as a macro if used often:
`#define P(x) reinterpret_cast<Pi *>(x)`

Working with arrays

- Copy values from JVM memory to C/C++ memory

```
jsize arr_len = env->GetArrayLength(columnIndexes);  
jlong *arr = env->GetLongArrayElements(columnIndexes, NULL);  
// use the elements of arr  
env->ReleaseLongArrayElements(columnIndexes, arr, 0);
```

- Remember to free C/C++ memory → avoid memory leaks

```
jArray = env->NewByteArray(jlen);  
if (jArray)  
    // Copy data to Byte[]  
    env->SetByteArrayRegion(jArray, 0, jlen,  
        reinterpret_cast<const jbyte*>(bufPtr));  
free(bufPtr);  
return jArray;
```

- Create a new Java array
- Copy C/C++ array to Java array

Strings as trouble-makers

- Java strings are in modified UTF-8
 - null character encoded as 0xC0 0x80 (not valid UTF-8)
 - Many C/C++ libraries assume plain UTF-8

Create a new Java string

```
jchar stack_buf[stack_buf_size];
// adding characters to stack_buf
jchar* out_begin = stack_buf;
if (int_cast_with_overflow_detect(out_curr - out_begin, out_size))
    throw runtime_error("String size overflow");
return env->NewString(out_begin, out_size);
```

- Copy Java string to C/C++ array
- Remember to deallocate C array

```
jstring s; // typical a JNI function parameter
jchar *c = e->GetStringChars(s, 0);
// use c
env->ReleaseStringChars(s, c);
```

Exercise 2

- Create a simple Android app
 - Implement Java classes RNG and Pi
- Generate header files
- Implement intermediate C++ function and call original C++ code
- Go to <https://github.com/kneth/AndroidCalcPi> if lazy 😊

C++ exceptions

- C++ code can throw exceptions
 - new can throw bad_alloc,etc.
- If uncaught the app crashes with hard-to-understand messages in the log

- Better solution:
 - catch and rethrow as Java exception

```
Build fingerprint: 'generic_x86/sdk_x86/generic_x86:4.4.2/KK/999428:eng/test-keys'
Revision: '0'
pid: 1890, tid: 1890, name: t.zigzak.calcpi >>> net.zigzak.calcpi <<<
signal 6 (SIGABRT), code -6 (SI_TKILL), fault addr -----
eax 00000000 ebx 00000762 ecx 00000762 edx 00000006
esi 00000762 edi 0000000b
xcs 00000073 xds 0000007b xes 0000007b xfs 00000000 xss 0000007b
eip b7707c96 ebp b776cce0 esp bfa22090 flags 00200203
backtrace:
#00 pc 0003bc96 /system/lib/libc.so (tgkill+22)
#01 pc 00000005 <unknown>
stack:
bfa22050 00000001
bfa22054 b899c6d0 [heap]
bfa22058 00000015
bfa2205c b76d9ef9 /system/lib/libc.so (pthread_mutex_unlock+25)
bfa22060 a73bfd19 /data/app-lib/net.zigzak.calcpi-1/libgnustl_shared.so
bfa22064 b7767fcc /system/lib/libc.so
```

Throwing a Java exception

- You don't throw an exception
- You set the “exception” state in JVM
- Return from C/C++ function



follow the instructions: hand werpen <http://bit.ly/1Bo3gZx>

1 Get a reference to the exception

```
try {  
    Pi *pi = new Pi(static_cast<long>(iterations));  
    return reinterpret_cast<jlong>(pi);  
}  
catch (std::exception& e) {  
    jclass jExceptionClass =  
        env->FindClass("java/lang/RuntimeException");  
    std::ostringstream message;  
    message << "Allocating native class Pi failed: " << e.what();  
    env->ThrowNew(jExceptionClass, message.str().c_str());  
    env->DeleteLocalRef(jExceptionClass);  
}  
return 0;
```

2 Set the “exception” state

3 Clean up

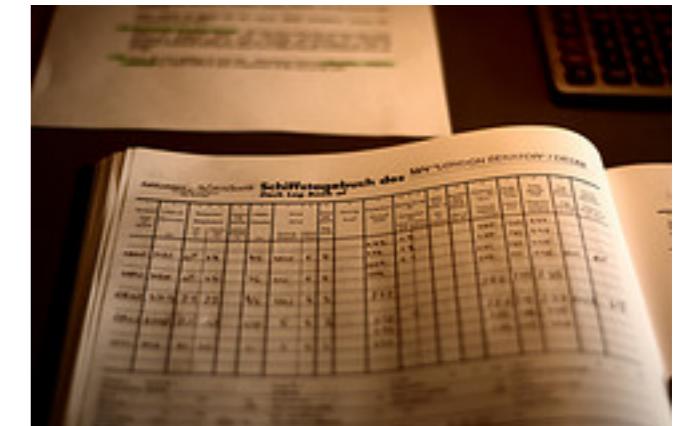
Logging

- JNI provides access to the Android log
- `__android_log_print` is a simple variadic log function
- Linking with `liblog` is required
- `size_t` depends on bit width → cast to `int64_t` before logging

Macro to simplify logging

```
#define LOG(...) __android_log_print(ANDROID_LOG_DEBUG, "calcPi", __VA_ARGS__);
```

```
android.ndk {  
    moduleName = "calcPi"  
    cppFlags += ["-fexceptions"]  
    ldLibs += ["log"]  
    stl = "gnustl_shared" // for exceptions  
}
```



vitelone: Deck Log Book <http://bit.ly/1AXKZ0j>

Link flag

Tracing JNI calls

- It is useful to trace calls
 - Define ENTER, ENTER_PTR, and LEAVE macros
 - Logging native pointer as jlong often produce negative numbers - don't worry

```
1955-1955/net.zigzak.calcpi D/calcPi : Enter Java_net_zigzak_calcpi_NativePi_nativeCreate
1955-1955/net.zigzak.calcpi D/calcPi : iterations: 10000
1955-1955/net.zigzak.calcpi D/calcPi : Enter Java_net_zigzak_calcpi_NativePi_nativeEstimate
-1202665872
1955-1955/net.zigzak.calcpi D/calcPi : estimate = 3.128400e+00
```

Exercise 3

- Catch common C++ exceptions (`bad_alloc`)
- Rethrow as a Java exception
- Adding logging/tracing of JNI calls
- Go to <https://github.com/kneth/AndroidCalcPi> if lazy 😊

Tricks to compiling
and linking

Reduce size

- Compiler can optimise for size (-Os)
- Link Time Optimizations (LTO) can reduce size of native code
- Eliminate unused code (-ffunction-sections -fdata-sections) and unused ELF symbols (-fvisibility=hidden)

```
android.ndk {  
    moduleName = "calcPi"  
    cppFlags += ["-Os", "-fexceptions", "-fvisibility=hidden"  
                "-ffunction-sections", "-fdata-sections",  
                "-Wl,--gc-sections", "-flto"]  
    ldLibs += "log"  
    stl = "gnustl_shared" // for exceptions  
    ldFlags += ["-Wl,--gc-sections", "-flto"] // reduce size  
}
```

Threading

POSIX threads

- Android's Bionic C library supports POSIX threads
 - except cancellation
- Threading can utilize your device's cores

Exercise 4

- Add a π estimator using POSIX threads
 - reuse the estimate method
- Go to <https://github.com/kneth/AndroidCalcPi> if lazy 😊

Using non-NDK libraries

Libraries

- NDK has some libraries
 - Basic ones: standard C, logging, etc.
 - Linked during build
- Android has many other libraries
 - /system/lib is their home
 - OpenSSL, XML2, SQLite, ICU

How to use?

- Android has dynamic linking
 - Add d1 to ldFlags
 - Include d1fcn.h in your C/C++ files

Find symbol
in .so file

- You might have to declare relevant structs and types

Function pointer

```
SHA256_CTX ctx;
int (*SHA224_Init) (SHA256_CTX *) ;
SHA224_Init =
    reinterpret_cast<int (*) (SHA256_CTX*)>
        (dlsym(RTLD_DEFAULT, "SHA224_Init"));
SHA224_Init(&ctx);
```

Exercise 5

- Add method to a class
 - Load one or two libcrypto functions
 - and call them
- Go to <https://github.com/kneth/AndroidCalcPi> if lazy 😊

Benchmarking

What makes benchmarking hard?



William Warby, Triple Timer; <http://bit.ly/1TwetP4>

- Android devices do many things
 - checking for updates, synchronising accounts, garbage collection, etc.
- Android timers are not reliable
- Just-in-Time compilation

Measuring timer resolution

- Use
Debug.threadCpuTimeNanos()
- But resolution depends on device
- Microbenchmark must be longer than resolution
- Resolution is used to validate benchmark

```
long diff = 50000;
for (int i = 0; i < 100; i++) {
    long end;
    long start =
        Debug.threadCpuTimeNanos();
    while (true) {
        end = Debug.threadCpuTimeNanos();
        if (end != start) {
            if (diff > (end - start)) {
                diff = end - start;
            }
            break;
        }
    }
}
```

Debugging

Android Debug Bridge

- Android Debug Bridge (adb) has some useful subcommands
 - shell: plain command-line shell
 - pull: copy file from device
 - push: copy file to device



M. O'Thompsonski, <http://bit.ly/1Da0o0h>

Get stack trace

- Android apps are processes
 - Dump stack traces using kill
 - Log every native call

```
mkdir -p /data/anr  
ps # find your app  
kill -3 <pid>  
cat /data/anr/traces.txt
```

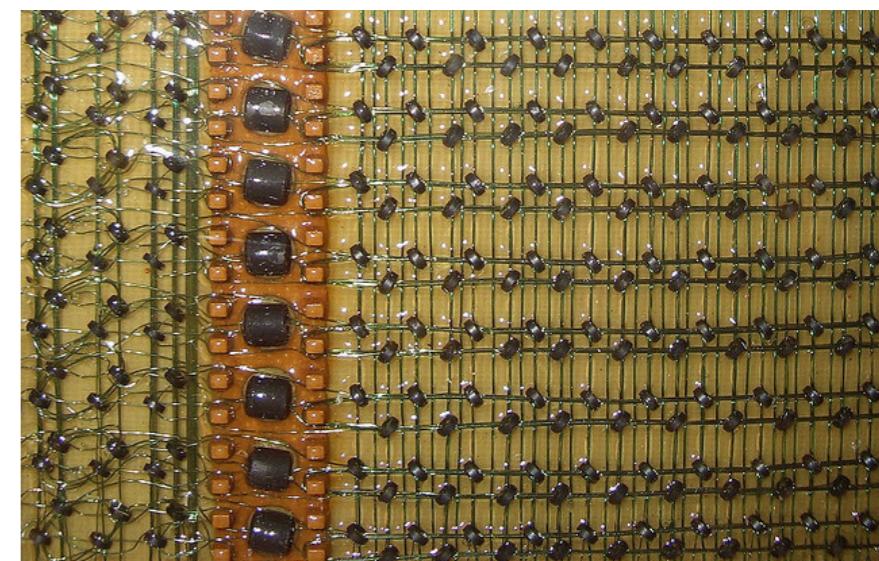
```
#define ENTER_PTR(ptr)  
__android_log_print(ANDROID_LOG_DEBUG, "calcPi", "Enter %s %" PRIId64,  
__FUNCTION__, static_cast<int64_t>(ptr));
```

Forces 64 bit value!

The name of the function

Checking native memory

- The emulator can help checking malloc
- C/C++ makes it easy to do memory management wrong
- Set the debug level to 20



Steve Jurvetson, Primitive Memories; <http://bit.ly/1HY80sf>

```
adb shell setprop libc.debug.malloc 20
adb shell stop
adb shell start
```

Native debugging

- Android Studio 1.3 supports native debugging
 - gdb or lldb
- Add a run configuration (native)

```
android.buildTypes {  
    debug {  
        isJniDebuggable = true  
        isDebuggable = true  
    }  
}
```

App's build.gradle

Exercise 6

- Add a native debugging configuration
- Set breakpoint in native function
- Check variables and single step

References

- *High Performance Android Apps* by Doug Sillars. O'Reilly Media, 2015.
- *Essential JNI* by Rob Gordon. Prentice Hall, 1998.
- *The Java Native Interface* by Sheng Liang. Addison-Wesley, 1999.
- *Android NDK* by Sylvian Rataboil. Packt Publishing, 2012.



AnDevCon

The Android Developer Conference

Please take a moment to fill out
the class feedback form via the
app. Paper feedback forms are
also available in the back of the
room.

eventmobi.com/adcboston