JNI

Karol Wrótniak

@DroidsOnRoids
@GDG Wrocław

karol.wrotniak@droidsonroids.pl

koral--

karol-wrotniak

@karol.wrotniak

Overview

- **Java Native Interface**
- Native:
  - Platform-specific shared library (.dll, .so, .dylib)
  - ~~Non-hybrid mobile app, React Native~~
  - C, C++
- Call native functions from Java and vice versa
- Other JVM languages supported
- JNI ≠ JNA

Foo.java:

```
static {
    System.loadLibrary("foo");
}

public native int getTextLength(String text);
```

Foo.java:

```
static {
    System.loadLibrary("foo");
}

public native int getTextLength(String text);
```

Foo.java:

```
static {
    System.loadLibrary("foo");
}

public native int getTextLength(String text);
```

foo.c:

```
#include <jni.h>

JNIEXPORT
jint
JNICALL Java_pl_droidsonroids_ndkdemo_Foo_getTextLength(
        JNIEnv *env,
        jobject thiz,
        jstring text
) {
    return (*env)->GetStringUTFLength(env, text);
}
```

Foo.java:

```
static {
    System.loadLibrary("foo");
}

public native int getTextLength(String text);
```

foo.c:

```
#include <jni.h>

JNIEXPORT
jint
JNICALL Java_pl_droidsonroids_ndkdemo_Foo_getTextLength(
        JNIEnv *env,
        jobject thiz,
        jstring text
) {
    return (*env)->GetStringUTFLength(env, text);
}
```

Foo.java:

```
static {
    System.loadLibrary("foo");
}

public native int getTextLength(String text);
```

foo.c:

```
#include <jni.h>

JNIEXPORT
jint
JNICALL Java_pl_droidsonroids_ndkdemo_Foo_getTextLength(
        JNIEnv *env,
        jobject thiz,
        jstring text
) {
    return (*env)->GetStringUTFLength(env, text);
}
```

Foo.java:

```
static {
    System.loadLibrary("foo");
}

public native int getTextLength(String text);
```

foo.c:

```
#include <jni.h>

JNIEXPORT
jint
JNICALL Java_pl_droidsonroids_ndkdemo_Foo_getTextLength(
        JNIEnv *env,
        jobject thiz,
        jstring text
) {
    return (*env)->GetStringUTFLength(env, text);
}
```

Foo.java:

```
static {
    System.loadLibrary("foo");
}

public native int getTextLength(String text);
```

foo.c:

```
#include <jni.h>

JNIEXPORT
jint
JNICALL Java_pl_droidsonroids_ndkdemo_Foo_getTextLength(
        JNIEnv *env,
        jobject thiz,
        jstring text
) {
    return (*env)->GetStringUTFLength(env, text);
}
```

Foo.java:

```
static {
    System.loadLibrary("foo");
}

public native int getTextLength(String text);
```

foo.c:

```
#include <jni.h>

JNIEXPORT
jint
JNICALL Java_pl_droidsonroids_ndkdemo_Foo_getTextLength(
        JNIEnv *env,
        jobject thiz,
        jstring text
) {
    return (*env)->GetStringUTFLength(env, text);
}
```

Foo.java:

```
static {
    System.loadLibrary("foo");
}

public native int getTextLength(String text);
```

foo.c:

```
#include <jni.h>

JNIEXPORT
jint
JNICALL Java_pl_droidsonroids_ndkdemo_Foo_getTextLength(
        JNIEnv *env,
        jobject thiz,
        jstring text
) {
    return (*env)->GetStringUTFLength(env, text);
}
```

Foo.java:

```
static {
    System.loadLibrary("foo");
}

public native int getTextLength(String text);
```

foo.c:

```
#include <jni.h>

JNIEXPORT
jint
JNICALL Java_pl_droidsonroids_ndkdemo_Foo_getTextLength(
        JNIEnv *env,
        jobject thiz,
        jstring text
) {
    return (*env)->GetStringUTFLength(env, text);
}
```

Foo.java:

```
static {
    System.loadLibrary("foo");
}

public native int getTextLength(String text);
```

foo.c:

```
#include <jni.h>

JNIEXPORT
jint
JNICALL Java_pl_droidsonroids_ndkdemo_Foo_getTextLength(
        JNIEnv *env,
        jobject thiz,
        jstring text
) {
    return (*env)->GetStringUTFLength(env, text);
}
```

Foo.java:

```
static {
    System.loadLibrary("foo");
}

public native int getTextLength(String text);
```

foo.c:

```
#include <jni.h>

JNIEXPORT
jint
JNICALL Java_pl_droidsonroids_ndkdemo_Foo_getTextLength(
        JNIEnv *env,
        jobject thiz,
        jstring text
) {
    return (*env)->GetStringUTFLength(env, text);
}
```

Why JNI?

- Reuse existing native libraries
- Hinder reverse-engineering
- Increase performance (**in certain cases only**)
  - computation
- Implement operations not available in Java:
  - System clock - `System.currentTimeMillis()`
  - Filesystem access
  - other OS-specific features

Sample projects using JNI

android-gif-drawable

Views and Drawable for displaying animated GIFs on Android

Java ★ 5.8k 1.4k

Java API

- `System.loadLibrary("foo")`
  - `System.mapLibraryName(“foo”) → "libfoo.so"`
  - `java.library.path` property
- `System.load("/usr/local/lib/libfoo.so")`

JNI types

- jbyte, jchar, jshort, jint, jlong, jfloat, jdouble
- jboolean: JNI_TRUE, JNI_FALSE
- jstring, jthrowable
- jarray, jintarray...
- jclass, jobject
- jmethodID, jfieldID

Calling Java from C

```
public static long min(long a, long b)
```

```
jclass mathClass = (*env)->FindClass(env, "java/lang/Math");

jmethodID minMethodID = (*env)->GetStaticMethodID(env, mathClass,
"min", "(JJ)J");

jlong min = (*env)->CallStaticLongMethod(env, mathClass,
minMethodID, x, y);
```

Calling Java from C

```
public static long min(long a, long b)
```

```
jclass mathClass = (*env)->FindClass(env, "java/lang/Math");

jmethodID minMethodID = (*env)->GetStaticMethodID(env, mathClass,
"min", "(JJ)J");

jlong min = (*env)->CallStaticLongMethod(env, mathClass,
minMethodID, x, y);
```

Calling Java from C

```
public static long min(long a, long b)
```

```
jclass mathClass = (*env)->FindClass(env, "java/lang/Math");

jmethodID minMethodID = (*env)->GetStaticMethodID(env, mathClass,
"min", "(JJ)J");

jlong min = (*env)->CallStaticLongMethod(env, mathClass,
minMethodID, x, y);
```

Calling Java from C

```
public static long min(long a, long b)
```

```
jclass mathClass = (*env)->FindClass(env, "java/lang/Math");

jmethodID minMethodID = (*env)->GetStaticMethodID(env, mathClass,
"min", "(JJ)J");

jlong min = (*env)->CallStaticLongMethod(env, mathClass,
minMethodID, x, y);
```

Calling Java methods

```
public static long min(long a, long b)
```

```
jclass mathClass = (*env)->FindClass(env, "java/lang/Math");

jmethodID minMethodID = (*env)->GetStaticMethodID(env, mathClass,
"min", "(JJ)J");

jlong min = (*env)->CallStaticLongMethod(env, mathClass,
minMethodID, x, y);
```

Accessing Java fields

```
jobject calendar = …

jclass calendar_class = (*env)->GetObjectClass(env, calendar);

jfieldID time_field_id = (*env)->GetFieldID(env, calendar_class,
                                            "time", "J");

jlong time = (*env)->GetLongField(env, calendar, time_field_id);
```

Creating Java objects

```
jclass foo_class = (*env)->FindClass(env,
                                     "pl/droidsonroids/ndkdemo/Foo");

jmethodID constructor_id = (*env)->GetMethodID(env, foo_class,
                                               "<init>", "()V");

jobject foo = (*env)->NewObject(env, foo_class, constructor_id);
```

Creating Java objects

```
jclass foo_class = (*env)->FindClass(env,
                                     "pl/droidsonroids/ndkdemo/Foo");

jmethodID constructor_id = (*env)->GetMethodID(env, foo_class,
                                               "<init>", "()V");

jobject foo = (*env)->NewObject(env, foo_class, constructor_id);
```

Creating Java objects

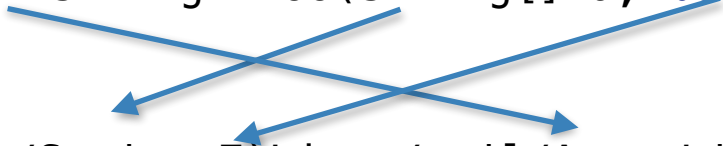
```
jobject foo = (*env)->AllocObject(env, foo_class);
```

Type signatures

Type signature	Java type	Example
Z	boolean	
B	byte	
C	char	
S	short	
I	int	
J	long	
F	float	
D	double	
L<FQCN>;	FQCN	Lcom/foo/Foo;
[<type>	type[]	[Z
V	void	

```
javap -s <class file>
```

```
abstract ArrayList<String> foo(String[] a, boolean b);
```



```
([Ljava/lang/String;Z)Ljava/util/ArrayList;
```

Reference types

- Local
- Global
- WeakGlobal (weaker than Java Weak and Soft)

Local references

```
jobject persistent_foo;

JNIEXPORT void JNICALL
Java_pl_droidsonroids_ndkdemo_Foo_foo(
        JNIEnv *env,
        jobject thiz,
        jobject foo
) {
    persistent_foo = foo;
    printf("foo");
}
```

Global References

```
jobject persistent_foo;

JNIEXPORT void JNICALL
Java_pl_droidsonroids_ndkdemo_Foo_foo(
        JNIEnv *env,
        jobject thiz,
        jobject foo
) {
    persistent_foo = (*env)->NewGlobalRef(env, foo);
    printf("foo");
}
```

```
(*env)->DeleteGlobalRef(env, persistent_foo);
```

Finalizers

```
public final class Foo {
    @Override
    protected void finalize() throws Throwable {
        releaseBar(bar);
        super.finalize();
    }

    private final long bar;

    public Foo() {
        bar = createBar();
    }

    private native long createBar();

    private native void releaseBar(long bar);
}
```

Finalizers

```
public final class Foo {
    @Override
    protected void finalize() throws Throwable {
        releaseBar(bar);
        super.finalize();
    }

    private final long bar;

    public Foo() {
        bar = createBar();
    }

    private native long createBar();

    private native void releaseBar(long bar);
}
```

Finalizers

```
public final class Foo {
    @Override
    protected void finalize() throws Throwable {
        releaseBar(bar);
        super.finalize();
    }

    private final long bar;

    public Foo() {
        bar = createBar();
    }

    private native long createBar();

    private native void releaseBar(long bar);
}
```

Finalizer guardian

```
public class Foo {

    @SuppressWarnings("unused")
    private final Object finalizer = new Object() {
        @Override
        protected void finalize() throws Throwable {
            releaseBar(bar);
        }
    };

    private final long bar;

    public Foo() {
        bar = createBar();
    }

    private native long createBar();
    private native void releaseBar(long bar);
}
```

JNI_OnLoad JNI_OnUnload

```
void *foo;
JavaVM *global_vm;

JNIEXPORT
jint JNICALL JNI_OnLoad(JavaVM *vm, void *reserved) {
    global_vm = vm;

    if (!init()) {
        return JNI_ERR;
    }

    foo = malloc(1);

    return JNI_VERSION_1_8;
}

JNIEXPORT void JNICALL JNI_OnUnload(JavaVM *vm, void *reserved) {
    free(foo);
}
```

JNI_OnLoad JNI_OnUnload

```
void *foo;
JavaVM *global_vm;

JNIEXPORT
jint JNICALL JNI_OnLoad(JavaVM *vm, void *reserved) {
    global_vm = vm;

    if (!init()) {
        return JNI_ERR;
    }

    foo = malloc(1);

    return JNI_VERSION_1_8;
}

JNIEXPORT void JNICALL JNI_OnUnload(JavaVM *vm, void *reserved) {
    free(foo);
}
```

JNI_OnLoad JNI_OnUnload

```
void *foo;
JavaVM *global_vm;

JNIEXPORT
jint JNICALL JNI_OnLoad(JavaVM *vm, void *reserved) {
    global_vm = vm;

    if (!init()) {
        return JNI_ERR;
    }

    foo = malloc(1);

    return JNI_VERSION_1_8;
}

JNIEXPORT void JNICALL JNI_OnUnload(JavaVM *vm, void *reserved) {
    free(foo);
}
```

JNI_OnLoad JNI_OnUnload

```
void *foo;
JavaVM *global_vm;

JNIEXPORT
jint JNICALL JNI_OnLoad(JavaVM *vm, void *reserved) {
    global_vm = vm;

    if (!init()) {
        return JNI_ERR;
    }

    foo = malloc(1);

    return JNI_VERSION_1_8;
}

JNIEXPORT void JNICALL JNI_OnUnload(JavaVM *vm, void *reserved) {
    free(foo);
}
```

Attaching to native threads

```
void *doSomething(void *args) {
    JavaVMAttachArgs attach_args = {
        .group = NULL,
        .name = "WorkerThread",
        .version = JNI_VERSION_1_6
    };

    JNIEnv *env;

    if ((*global_vm)->AttachCurrentThread(global_vm, &env,
                                          &attach_args) == JNI_OK) {
        //JNI function calls
        (*global_vm)->DetachCurrentThread(global_vm);
    }
}
```

Attaching to native threads

```
void *doSomething(void *args) {
    JavaVMAttachArgs attach_args = {
        .group = NULL,
        .name = "WorkerThread",
        .version = JNI_VERSION_1_6
    };

    JNIEnv *env;

    if ((*global_vm)->AttachCurrentThread(global_vm, &env,
                                          &attach_args) == JNI_OK) {
        //JNI function calls
        (*global_vm)->DetachCurrentThread(global_vm);
    }
}
```

Attaching to native threads

```
void *doSomething(void *args) {
    JavaVMAttachArgs attach_args = {
        .group = NULL,
        .name = "WorkerThread",
        .version = JNI_VERSION_1_6
    };

    JNIEnv *env;

    if ((*global_vm)->AttachCurrentThread(global_vm, &env,
                                          &attach_args) == JNI_OK) {
        //JNI function calls
        (*global_vm)->DetachCurrentThread(global_vm);
    }
}
```

- `AttachCurrentThreadAsDaemon`
- Detach using `pthread_key_create` destructor

Registering native methods

```java
//Foo.java
public native int plus(int a, int b);

//Foo.c
static jint add(JNIEnv *env, jobject thiz, jint a, jint b) {
    return a + b;
}
```

Registering native methods

```
//Foo.java
public native int plus(int a, int b);

//Foo.c
static jint add(JNIEnv *env, jobject thiz, jint a, jint b) {
    return a + b;
}

JNIEXPORT
jint JNICALL JNI_OnLoad(JavaVM *vm, void *reserved) {

    JNINativeMethod methods[] = {
            {"plus",  "(II)I", (void *) add},
    };

    return JNI_VERSION_1_6;
}
```

Registering native methods

```
//Foo.java
public native int plus(int a, int b);

//Foo.c
static jint add(JNIEnv *env, jobject thiz, jint a, jint b) {
    return a + b;
}

JNIEXPORT
jint JNICALL JNI_OnLoad(JavaVM *vm, void *reserved) {

    JNINativeMethod methods[] = {
            {"plus",  "(II)I", (void *) add}
    };

    JNIEnv *env;
    (*vm)->GetEnv(vm, (void **) &env, JNI_VERSION_1_6);

    jclass foo_class = (*env)->FindClass(env,
                                         "pl/droidsonroids/ndkdemo/Foo");
    (*env)->RegisterNatives(env, foo_class, methods, 1);

    return JNI_VERSION_1_6;
}
```

Handling exceptions

- C++ exceptions are **not** converted to Java counterparts
- JNI functions:
  - ExceptionCheck
  - ExceptionOccured
  - ExceptionClear
  - FatalError
  - ExceptionDescribe

Catching exceptions

```
(*env)->CallVoidMethod(env, foo, fooMethodID);
if ((*env)->ExceptionCheck(env) == JNI_TRUE) {
    (*env)->ExceptionClear(env);
    (*env)->CallVoidMethod(env, foo, barMethodID);
}
```

catch

- Need to clear exception or return before JNI calls

Throwing Exceptions

```
jclass exceptionClass = (*env)->FindClass(env, "android/system/
ErrnoException");

jmethodID const constructorID = (*env)->GetMethodID(env,
exceptionClass, "<init>", "(Ljava/lang/String;I)V");

jobject exception = (*env)->NewObject(env, exceptionClass,
constructorID, number);

(*env)->Throw(env, exception);
```

Throwing Exceptions

```
jclass exceptionClass = ...

(*env)->ThrowNew(env, exceptionClass, "Foo failed");
```

Arrays

```
JNIEXPORT void JNICALL
Java_pl_droidsonroids_ndkdemo_Foo_foo(JNIEnv *env, jobject thiz,
                                      jintArray points) {
    jsize length = (*env)->GetArrayLength(env, points);
    jint *native_points = (*env)->GetIntArrayElements(env, points,
                                                      NULL);
    //...
    (*env)->ReleaseIntArrayElements(env, points, native_points, 0);
}
```

Arrays

```
JNIEXPORT void JNICALL
Java_pl_droidsonroids_ndkdemo_Foo_foo(JNIEnv *env, jobject thiz,
                                      jintArray points) {
    jsize length = (*env)->GetArrayLength(env, points);
    jint *native_points = (*env)->GetIntArrayElements(env, points,
                                                      NULL);
    //...
    (*env)->ReleaseIntArrayElements(env, points, native_points, 0);
}
```

Arrays

```
JNIEXPORT void JNICALL
Java_pl_droidsonroids_ndkdemo_Foo_foo(JNIEnv *env, jobject thiz,
                                      jintArray points) {
    jsize length = (*env)->GetArrayLength(env, points);
    jint *native_points = (*env)->GetIntArrayElements(env, points,
                                                      NULL);
    //...
    (*env)->ReleaseIntArrayElements(env, points, native_points, 0);
}
```

Release mode

Arrays

```
JNIEXPORT void JNICALL
Java_pl_droidsonroids_ndkdemo_Foo_foo(JNIEnv *env, jobject thiz,
                                      jintArray points) {
    jsize length = (*env)->GetArrayLength(env, points);
    jint *native_points = (*env)->GetIntArrayElements(env, points,
                                                      NULL); *isCopy
    //...
    (*env)->ReleaseIntArrayElements(env, points, native_points, 0);
}
```

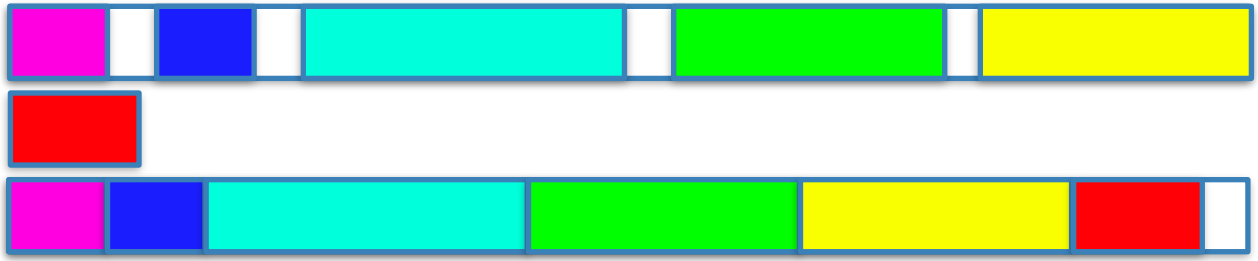
Arrays

```
JNIEXPORT void JNICALL
Java_pl_droidsonroids_ndkdemo_Foo_foo(JNIEnv *env, jobject thiz,
                                      jintArray points) {
    jsize length = (*env)->GetArrayLength(env, points);
    jint *native_points = (*env)->GetIntArrayElements(env, points,
                                                      NULL);
    //...
    (*env)->ReleaseIntArrayElements(env, points, native_points, 0);
}

JNIEXPORT void JNICALL
Java_pl_droidsonroids_ndkdemo_Foo_foo(JNIEnv *env, jobject thiz, jintArray
                                      points) {
    jsize start = 0;
    jsize size = 2;
    jint native_points[2];
    (*env)->GetIntArrayRegion(env, points, start, size, native_points);
    native_points[0] = native_points[1];
    (*env)->SetIntArrayRegion(env, points, start, size, native_points);
}
```

Arrays

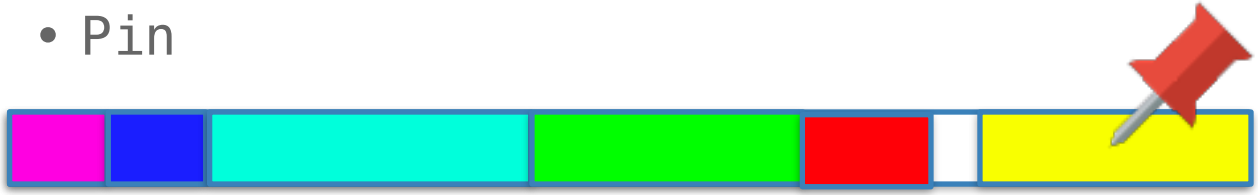
```
JNIEXPORT void JNICALL
Java_pl_droidsonroids_ndkdemo_Foo_foo(JNIEnv *env, jobject thiz,
                                      jintArray points) {
    jsize length = (*env)->GetArrayLength(env, points);
    jint *native_points = (*env)->GetIntArrayElements(env, points,
                                                      NULL);
    //...
    (*env)->ReleaseIntArrayElements(env, points, native_points, 0);
}

JNIEXPORT void JNICALL
Java_pl_droidsonroids_ndkdemo_Foo_foo(JNIEnv *env, jobject thiz, jintArray
                                      points) {
    jsize start = 0;
    jsize size = 2;
    jint native_points[2];
    (*env)->GetIntArrayRegion(env, points, start, size, native_points);
    native_points[0] = native_points[1];
    (*env)->SetIntArrayRegion(env, points, start, size, native_points);
}
```

GC compaction



- Copy

- Disable GC between Get and Release
- Pin



Release modes & isCopy

Mode	Copy back	Free buffer
0	✓	✓
JNI_COMMIT	✓	✗
JNI_ABORT	✗	✓

Need to release with another mode

`*isCopy == JNI_FALSE:`

- skip useless commit
- restore original contents before abort

Direct byte buffers

```
JNIEXPORT
jobject JNICALL Java_pl_droidsonroids_ndkdemo_Foo_foo(
        JNIEnv *env,
        jobject thiz,
        jobject buffer
) {
    void *foo = (*env)->GetDirectBufferAddress(env, buffer);
    jlong capacity = (*env)->GetDirectBufferCapacity(env,
                                                     buffer);
    //...
}
```

Direct byte buffers

```
JNIEXPORT
jobject JNICALL Java_pl_droidsonroids_ndkdemo_Foo_foo(
        JNIEnv *env,
        jobject thiz,
        jobject buffer
) {
    void *foo = (*env)->GetDirectBufferAddress(env, buffer);
    jlong capacity = (*env)->GetDirectBufferCapacity(env,
                                                     buffer);

    //...
}
```

Direct byte buffers

```
JNIEXPORT
jobject JNICALL Java_pl_droidsonroids_ndkdemo_Foo_foo(
        JNIEnv *env,
        jobject thiz,
        jobject buffer
) {
    void* bar = mmap(…
    return (*env)->NewDirectByteBuffer(env, bar, capacity);
}
```

- underlying buffer has to be released

Modified UTF-8

- Null-terminated
- No null byte inside
- `U+0000 -> 0xC0 0x80`
- 4-byte UTF-8 format encoded using 2 x 3-bytes
- JNI functions:
  - *StringUTF* - modified UTF-8
    - native string APIs
  - *String* - standard UTF-8
    - raw bytes from native sources

Synchronization

```
(*env)->MonitorEnter(env, thiz);

//JNI function calls

(*env)->MonitorExit(env, thiz);


//or
(*global_vm)->DetachCurrentThread(global_vm);
```

JNA - Java Native Access

```
double modf(double x, double *intptr);
```

```
public interface CLibrary extends Library {
    CLibrary INSTANCE = (CLibrary)
            Native.loadLibrary((Platform.isWindows() ?
                                "msvcrt" : "c"),
                    CLibrary.class);

    double modf(double x, DoubleByReference intptr);
}
```

```
DoubleByReference intptrReference = new DoubleByReference();
double fraction = CLibrary.INSTANCE.modf(1.234, intptrReference);

System.out.println(fraction);
System.out.println(intptrReference.getValue());
```

JNA - Java Native Access

```
double modf(double x, double *intptr);
```

```
public interface CLibrary extends Library {
    CLibrary INSTANCE = (CLibrary)
            Native.loadLibrary((Platform.isWindows() ?
                                "msvcrt" : "c"),
                    CLibrary.class);

    double modf(double x, DoubleByReference intptr);
}
```

```
DoubleByReference intptrReference = new DoubleByReference();
double fraction = CLibrary.INSTANCE.modf(1.234, intptrReference);

System.out.println(fraction);
System.out.println(intptrReference.getValue());
```

JNA - Java Native Access

```
double modf(double x, double *intptr);
```

```
public interface CLibrary extends Library {
    CLibrary INSTANCE = (CLibrary)
            Native.loadLibrary((Platform.isWindows() ?
                                "msvcrt" : "c"),
                    CLibrary.class);

    double modf(double x, DoubleByReference intptr);
}
```

```
DoubleByReference intptrReference = new DoubleByReference();
double fraction = CLibrary.INSTANCE.modf(1.234, intptrReference);

System.out.println(fraction);
System.out.println(intptrReference.getValue());
```

JNA - Java Native Access

```
double modf(double x, double *intptr);
```

```
public interface CLibrary extends Library {
    CLibrary INSTANCE = (CLibrary)
            Native.loadLibrary((Platform.isWindows() ?
                                "msvcrt" : "c"),
                    CLibrary.class);

    double modf(double x, DoubleByReference intptr);
}
```

```
DoubleByReference intptrReference = new DoubleByReference();
double fraction = CLibrary.INSTANCE.modf(1.234, intptrReference);

System.out.println(fraction);
System.out.println(intptrReference.getValue());
```

Android POSIX API

android.system.Os

android.system.OsConstants

static FileDescriptor	accept(FileDescriptor fd, InetSocketAddress peerAddress) See accept(2).
static void	chown(String path, int uid, int gid) See chown(2).

public static final int	SIGSTOP
static boolean	WIFEXITED(int status) Tests whether the child exited normally.

Traps

- Signature inconsistency
- Exempt from obfuscation:
  - Native methods
  - Everything called from native
- Leaking `this` in `finalize`

Incomplete initialization

```
class Foo {
    @NonNull private final File file;
    private final FileOutputStream stream;

    Foo(@NonNull File file) throws IOException {
        stream = new FileOutputStream(file);
        stream.write(0);
        this.file = file;
    }

    @Override
    protected void finalize() throws Throwable {
        System.out.println("Closing " + file.getPath());
        try {
            stream.close();
        } finally {
            super.finalize();
        }
    }
}
```

IOException

NullPointerException

References

1. Java Native Interface Specification
2. JNI Tips
3. The Java™ Native Interface Programmer's Guide and Specification

Q&A