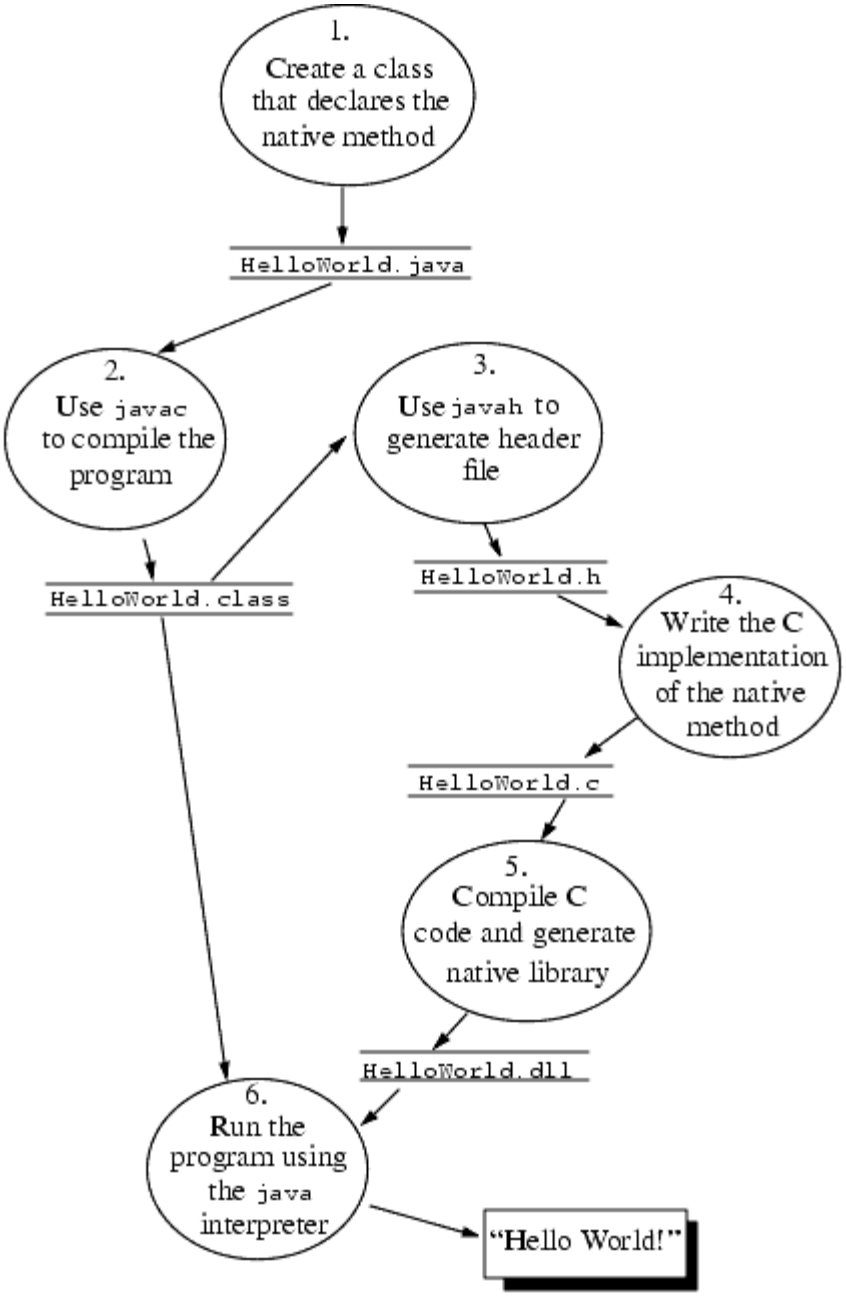
JNI

Tomasz Kubik


1.
Create a class that declares the native method
HelloWorld.java
2.
Use javac to compile the program
HelloWorld.class
3.
Use javah to generate header file
HelloWorld.h
4.
Write the C implementation of the native method
HelloWorld.c
5.
Compile C code and generate native library
HelloWorld.dll
6.
Run the program using the java interpreter
"Hello World!"

Krok 1: Kod JAVA

- ***HelloWorld.java***

```
class HelloWorld {
public native void displayMessage();
static {
  System.loadLibrary("HelloWorldImp"); } }
```

- ***Main.java***

```
class Main {
public static void main(String[] args) {
HelloWorld hello = new HelloWorld();
hello.displayMessage(); } }
```

System.load oraz System.loadLibrary

- System.load
  - argumentem jest nazwa biblioteki z pełną ścieżką
    - *.dll dla windows
    - *.so dla Linux lub Solaris
    - C:\dlls\myjni.dll
- System.loadLibrary
  - argumentem jest nazwa biblioteki (bez ścieżki i rozszerzenia, które dodawane są automatycznie)
  - ścieżka zapisana jest we własności systemowej, którą można „zobaczyć” w następujący sposób:

```
System.getProperty(
  "java.library.path" );
```

Krok 2: Kompilacja kodu JAVA

- javac HelloWorld.java
- javac Main.java

Krok 3: Generacja nagłówka

- Kiedyś kompilacja za pomocą javah ze skompilowanej klasy:
  - javah -jni HelloWorld
- Teraz kompilacja za pomocą javac ze źródeł:
  - javac -h c -d target java/HelloWorld.java

```
#include <jni.h>
#ifndef _Included_HelloWorld
#define _Included_HelloWorld
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     HelloWorld
 * Method:    displayMessage
 * Signature: ()V
 */
JNIEXPORT void JNICALL Java_HelloWorld_displayMessage
  (JNIEnv *, jobject);

#ifdef __cplusplus
}
#endif
#endif
```

Krok 4: Generacja kodu C++

- ***HelloWorld.cpp***

```
#include <stdio.h>
#include "HelloWorld.h"
JNIEXPORT void JNICALL
Java_HelloWorld_displayMessage(JNIEnv *env,
  jobject obj) {
printf("Hello World!\n");
}
```

Krok 5: Utworzenie bibliotek

- Kompilator g++

```
g++ -G -I/pkgs/jdk1.1.1/include -
I/pkgs/jdk1.1.1/include/solaris
HelloWorld.C -o libHelloWorldImp.so
```

- kompilator pod Windows

```
cl -Im:\jdk1.1.5\include -
Im:\jdk1.1.5\include\win32 -LD
HelloWorld.cpp -Fe HelloWorldImp.dll
```

Uruchomienie programu

- Bez parametrów wywołania

  > java Main

  Hello World!

- Z parametrami wywołania

  > java -Djava.library.path=. Main

  setenv LD_LIBRARY_PATH .

  LD_LIBRARY_PATH=.
  export LD_LIBRARY_PATH

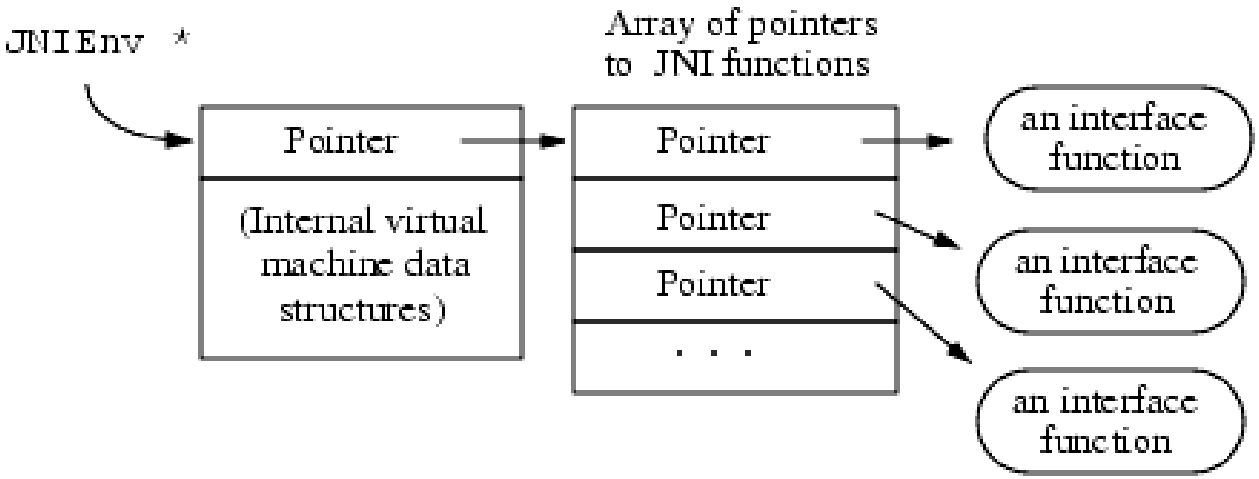
Uwagi na temat kodu C/C++

- **C syntax:**

```
jsize len =
(*env)->GetArrayLength(env,array);
```

- **C++ syntax:**

```
jsize len =
env->GetArrayLength(array);
```



Mapowanie typów

Typ natywny	Typ Java	Opis	Sygnatura typu
`unsigned char`	`jboolean`	`unsigned 8 bits`	`Z`
`signed char`	`jbyte`	`signed 8 bits`	`B`
`unsigned short`	`jchar`	`unsigned 16 bits`	`C`
`short`	`jshort`	`signed 16 bits`	`S`
`long`	`jint`	`signed 32 bits`	`I`
`long long`	`jlong`	`signed 64 bits`	`J`
`__int64`			
`float`	`jfloat`	`32 bits`	`F`
`double`	`jdouble`	`64 bits`	`D`

Typy podstawowe JNI używane w C/C++ (1)

Typ	Przeznaczenie	Typedef	Java/ C/C++
jboolean	8-bit boolean	typedef unsigned char jboolean;	C/C++
char	8-bit byte		C/C++
jbyte	8-bit signed byte	typedef signed char jbyte;	C/C++
WORD	16-bit unsigned short,	typedef unsigned short WORD;	C/C++
wchar_t	16-bit unsigned char		C/C++
WCHAR	16-bit unsigned char, wide char	typedef wchar_t WCHAR;	C/C++
TCHAR	16-bit unsigned char on Unicode supporting systems, 8-bit unsigned char or older systems, text char	typedef WCHAR TCHAR;	C/C++
jchar	16-bit unsigned char	typedef unsigned short jchar;	Java
jshort	16-bit signed short	typedef short jshort;	Java
DWORD	32-bit unsigned int, double word	typedef unsigned long DWORD;	C/C++
LPDWORD	DWORD *, pointer to 32-bit unsigned int, long pointer double word	typedef DWORD far * LPDWORD;	C/C++
BOOL	32-bit boolean. TRUE/true=1 FALSE/false=0, boolean	typedef int BOOL;	C/C++

Typy podstawowe JNI używane w C/C++ (2)

int	32-bit signed int		C/C++
long	32-bit signed int		C/C++
jint	32-bit signed int	typedef long jint;	Java
jfloat	32-bit signed float	typedef float jfloat;	Java
long long	64-bit signed long		C/C++
jlong	64-bit signed long	typedef __int64 jlong;	Java
LPSTR	char *, pointer to 8-bit null-terminated string	typedef __nullterminated CHAR * LPSTR;	C/C++
LPCSTR	const char *, constant pointer to 8-bit null-terminated string, long pointer constant string	typedef __nullterminated CONST CHAR * LPCSTR;	C/C++
LPTSTR	wchar_t *, pointer to 16-bit null-terminated string on Unicode-supporting platforms. On older platforms it means char *, pointer to 8-bit null-terminated string, long pointer text string	typedef LPCWSTR LPCTSTR;	C/C++
LPCTSTR	const wchar_t *, constant pointer to 16-bit null-terminated string on Unicode-supporting platforms. On older platforms it means char *, pointer to 8-bit null-terminated string, long pointer constant text string	typedef LPCWSTR LPCTSTR;	C/C++
jdouble	64-bit signed double	typedef double jdouble;	Java
jstring	counted Unicode String		Java
JNIEnv	the JNI environment with hooks to the JNI library methods		Java

Dostęp do obiektów String z kodu natywnego

```
JNIEXPORT jstring JNICALL
 Java_Prompt_getLine(JNIEnv *env, jobject obj, jstring prompt)
 {
     char buf[128];
     const jbyte *str;
     str = (*env)->GetStringUTFChars(env, prompt, NULL);
     if (str == NULL) {
         return NULL; /* OutOfMemoryError already thrown */
     }
     printf("%s", str);
     (*env)->ReleaseStringUTFChars(env, prompt, str);
     /* We assume here that the user does not type more than
      * 127 characters */
     scanf("%s", buf);
     return (*env)->NewStringUTF(env, buf);
 }
```

Zarządzanie pamięcią przy ciągach znaków (1)

```
const jchar *
  GetStringChars(JNIEnv *env,
   jstring str, jboolean *isCopy);
```

- w zależności od wartości ustawianej w `isCopy` (`JNI_TRUE`, `JNI_FALSE`) należy albo nie należy zwalniać uzyskanego ciągu znaków w kodzie natywnym

`ReleaseStringChars`

https://stackoverflow.com/questions/5859673/should-you-call-releasestringutfchars-if-getstringutfchars-returned-a-copy

http://barbie.uta.edu/~jli/Resources/Resource%20Provisoning&Performance%20Evaluation/85.pdf

Zarządzanie pamięcią przy ciągach znaków (2)

- `Get/ReleaseStringCritical`
  - pomiędzy parą tych funkcji kod natywny nie może zawierać blokujących wywołań albo rezerwować nowe obiekty na JVM

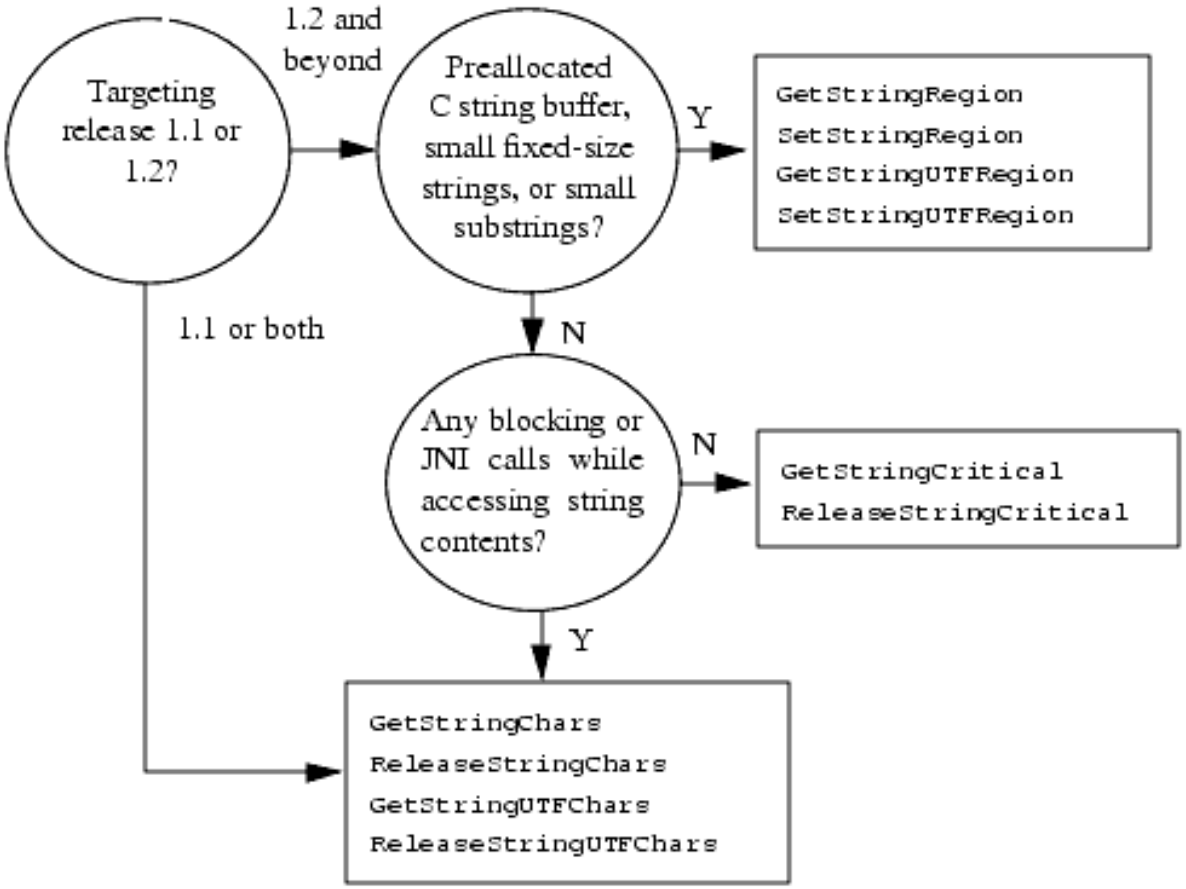
```
jchar *s1, *s2;
 s1 = (*env)->GetStringCritical(env, jstr1);
 if (s1 == NULL) {
     ... /* error handling */
 }
 s2 = (*env)->GetStringCritical(env, jstr2);
 if (s2 == NULL) {
     (*env)->ReleaseStringCritical(env, jstr1, s1);
     ... /* error handling */
 }
 ...     /* use s1 and s2 */
 (*env)->ReleaseStringCritical(env, jstr1, s1);
 (*env)->ReleaseStringCritical(env, jstr2, s2);
```

Zarządzanie pamięcią przy ciągach znaków (3)

```
JNIEXPORT jstring JNICALL
 Java_Prompt_getLine(JNIEnv *env, jobject obj, jstring
  prompt)
 {
     /* assume the prompt string and user input has less
  than 128
        characters */
    char outbuf[128], inbuf[128];
    int len = (*env)->GetStringLength(env, prompt);
    (*env)->GetStringUTFRegion(env, prompt, 0, len,
  outbuf);
    printf("%s", outbuf);
    scanf("%s", inbuf);
    return (*env)->NewStringUTF(env, inbuf);
 }
```

JNI Function	Description	Since
`GetStringChars` `ReleaseStringChars`	Obtains or releases a pointer to the contents of a string in Unicode format. May return a copy of the string.	JDK1.1
`GetStringUTFChars` `ReleaseStringUTFChars`	Obtains or releases a pointer to the contents of a string in UTF-8 format. May return a copy of the string.	JDK1.1
`GetStringLength`	Returns the number of Unicode characters in the string.	JDK1.1
`GetStringUTFLength`	Returns the number of bytes needed (not including the trailing 0) to represent a string in the UTF-8 format.	JDK1.1
`NewString`	Creates a java.lang.String instance that contains the same sequence of characters as the given Unicode C string.	JDK1.1
`NewStringUTF`	Creates a java.lang.String instance that contains the same sequence of characters as the given UTF-8 encoded C string.	JDK1.1
`GetStringCritical` `ReleaseStringCritical`	Obtains a pointer to the contents of a string in Unicode format. May return a copy of the string. Native code must not block between a pair of Get/ReleaseStringCritical calls.	Java 2 SDK1.2
`GetStringRegion` `SetStringRegion`	Copies the contents of a string to or from a preallocated C buffer in the Unicode format.	Java 2 SDK1.2
`GetStringUTFRegion` `SetStringUTFRegion`	Copies the content of a string to or from a preallocated C buffer in the UTF-8 format.	Java 2 SDK1.2

Dobór funkcji operujących na ciągach znaków



Dostęp do tablic z poziomu kodu natywnego (1)

```
class IntArray {
    private native int sumArray(int[] arr);
    public static void main(String[] args) {
        IntArray p = new IntArray();
        int arr[] = new int[10];
        for (int i = 0; i < 10; i++) {
            arr[i] = i;
        }
        int sum = p.sumArray(arr);
        System.out.println("sum = " + sum);
    }
    static {
        System.loadLibrary("IntArray");
    }
}
```

Dostęp do tablic z poziomu kodu natywnego (2)

- Kod C

```
JNIEXPORT jint JNICALL
 Java_IntArray_sumArray(JNIEnv *env, jobject obj,
  jintArray arr)
 {
     jint buf[10];
     jint i, sum = 0;
     (*env)->GetIntArrayRegion(env, arr, 0, 10,
  buf);
     for (i = 0; i < 10; i++) {
         sum += buf[i];
     }
     return sum;
 }
```

Dostęp do tablic z poziomu kodu natywnego (3)

```
JNIEXPORT jint JNICALL
 Java_IntArray_sumArray(JNIEnv *env, jobject obj,
   jintArray arr)
 {
     jint *carr;
     jint i, sum = 0;
     carr = (*env)->GetIntArrayElements(env, arr, NULL);
     if (carr == NULL) {
         return 0; /* exception occurred */
     }
     for (i=0; i<10; i++) {
         sum += carr[i];
     }
     (*env)->ReleaseIntArrayElements(env, arr, carr, 0);
     return sum;
 }
```

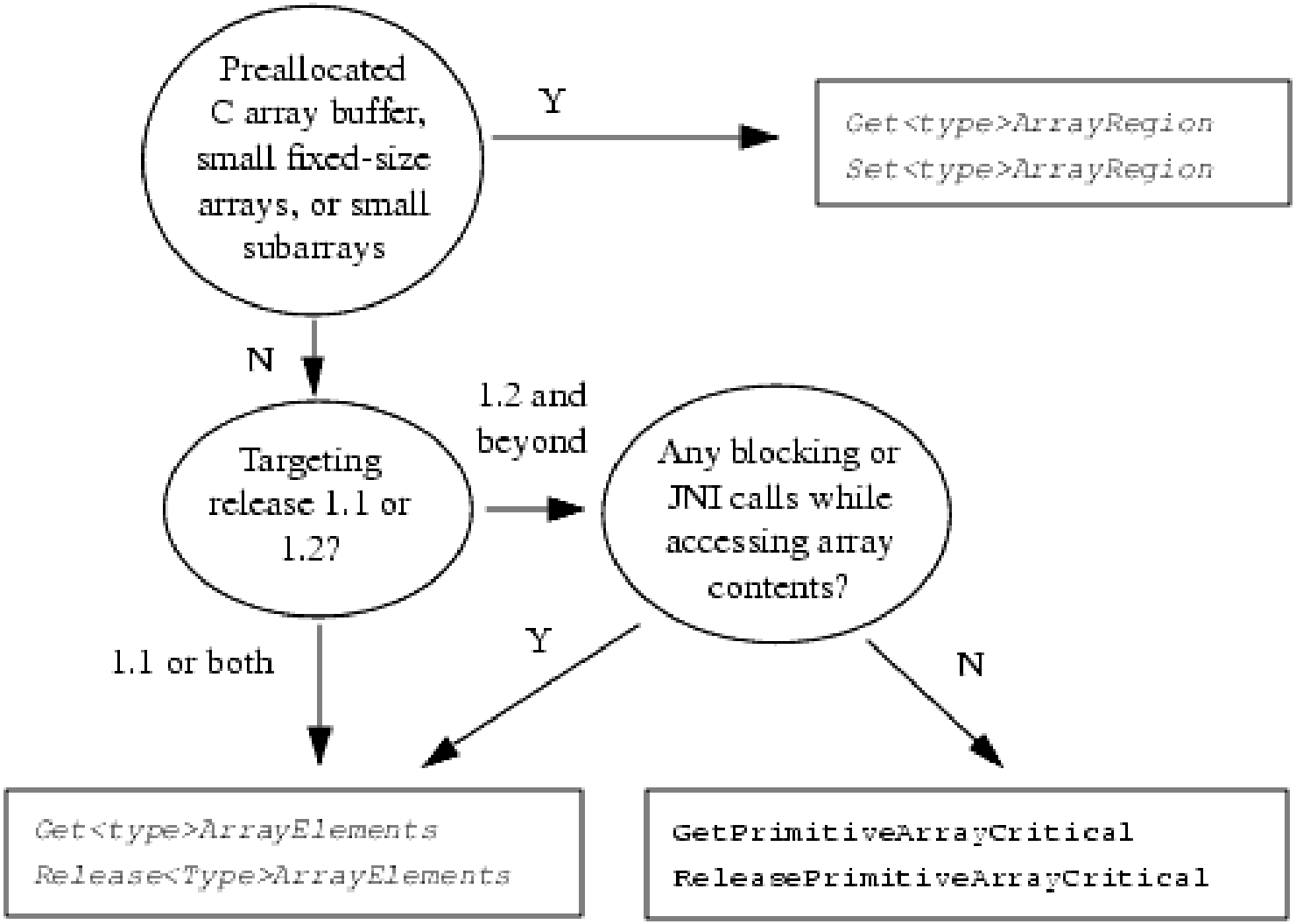
JNI Function	Description	Since
Get<Type>ArrayRegion Set<Type>ArrayRegion	Copies the contents of primitive arrays to or from a pre-allocated C buffer.	JDK1.1
Get<Type>ArrayElements Release<Type>ArrayElements	Obtains a pointer to the contents of a primitive array. May return a copy of the array.	JDK1.1
`GetArrayLength`	Returns the number of elements in the array.	JDK1.1
New<Type>Array	Creates an array with the given length.	JDK1.1
`GetPrimitiveArrayCritical` `ReleasePrimitiveArrayCritical`	Obtains or releases a pointer to the contents of a primitive array. May disable garbage collection, or return a copy of the array.	Java 2 SDK1.2

Funkcje manipulujące JNI (1)

Type	Get Parm	Put Parm	Release Parm	Return
Unicode String converted to 16-bit chars	GetStringChars GetStringLength	-	ReleaseStringChars	NewString
UTF-8 String converted to 8-bit chars	GetStringUTFChars GetStringLength	-	ReleaseStringUTFChars	NewStringUTF
int	use parm directly	-	-	use local jint
int[] copy access	GetIntArrayRegion GetArrayLength	SetIntArrayRegion	-	NewIntArray
int[] direct access	GetIntArrayElements GetArrayLength	-	ReleaseIntArrayElements	NewIntArray
Object	use parm directly	-	-	NewObject NewObjectA NewObjectV
Object[] copy access	GetObjectRegion GetArrayLength	SetObjectArrayRegion	-	NewObjectArray
Object[] direct access	GetObjectArrayElements GetArrayLength	-	ReleaseObjectArrayElements	NewObjectArray

Funkcje manipulujące JNI (2)

static field in Object	GetStaticFieldID	setStaticObjectField		
	GetStaticObjectField	SetStaticIntField		
	GetStringUTFChars			
	GetStaticIntField		-	-
instance field in Object	GetFieldID	SetObjectField		
	GetObjectField	SetIntField		
	GetIntField		-	-
callback static method	FindClass			
	GetStaticMethodID			
	CallStaticVoidMethod			
	CallStaticIntMethod			
	CallStaticObjectMethod	-	-	-
callback instance method	FindClass			
	GetMethodID			
	CallVoidMethod			
	CallIntMethod			
	CallObjectMethod	-	-	-


Preallocated C array buffer, small fixed-size arrays, or small subarrays
Y
Get<type>ArrayRegion
Set<type>ArrayRegion
N
Targeting release 1.1 or 1.2?
1.2 and beyond
Any blocking or JNI calls while accessing array contents?
1.1 or both
Y
N
Get<type>ArrayElements
Release<Type>ArrayElements
GetPrimitiveArrayCritical
ReleasePrimitiveArrayCritical

Tablice obiektów

```
class ObjectArrayTest {
    private static native int[][] initInt2DArray(int
  size);
    public static void main(String[] args) {
        int[][] i2arr = initInt2DArray(3);
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                System.out.print(" " + i2arr[i][j]);
            }
            System.out.println();
        }
    }
    static {
        System.loadLibrary("ObjectArrayTest");
    }
}
```

```
JNIEXPORT jobjectArray JNICALL
 Java_ObjectArrayTest_initInt2DArray(JNIEnv *env, jclass cls, int size){
     jobjectArray result;
     int i;
     jclass intArrCls = (*env)->FindClass(env, "[I");
     if (intArrCls == NULL) {
         return NULL; /* exception thrown */
     }
     result = (*env)->NewObjectArray(env, size, intArrCls, NULL);
     if (result == NULL) {
         return NULL; /* out of memory error thrown */
     }
     for (i = 0; i < size; i++) {
         jint tmp[256];  /* make sure it is large enough! */
         int j;
         jintArray iarr = (*env)->NewIntArray(env, size);
         if (iarr == NULL) {
             return NULL; /* out of memory error thrown */
         }
         for (j = 0; j < size; j++) {
             tmp[j] = i + j;
         }
         (*env)->SetIntArrayRegion(env, iarr, 0, size, tmp);
         (*env)->SetObjectArrayElement(env, result, i, iarr);
         (*env)->DeleteLocalRef(env, iarr);
     }
     return result;
 }
```

Dostęp do pól

```
class InstanceFieldAccess {
    private String s;

    private native void accessField();
    public static void main(String args[]) {
        InstanceFieldAccess c = new InstanceFieldAccess();
        c.s = "abc";
        c.accessField();
        System.out.println("In Java:");
        System.out.println("  c.s = \"" + c.s + "\"");
    }
    static {
        System.loadLibrary("InstanceFieldAccess");
    }
}
```

```
JNIEXPORT void JNICALL
 Java_InstanceFieldAccess_accessField(JNIEnv *env, jobject obj)
 {
     jfieldID fid;   /* store the field ID */
     jstring jstr;
     const char *str;

     jclass cls = (*env)->GetObjectClass(env, obj);
     printf("In C:\n");
     fid = (*env)->GetFieldID(env, cls, "s", "Ljava/lang/String;");
     if (fid == NULL) { return; /* failed to find the field */
     }
     jstr = (*env)->GetObjectField(env, obj, fid);
     str = (*env)->GetStringUTFChars(env, jstr, NULL);
     if (str == NULL) { return; /* out of memory */
     }
     printf("  c.s = \"%s\"\n", str);
     (*env)->ReleaseStringUTFChars(env, jstr, str);

     jstr = (*env)->NewStringUTF(env, "123");
     if (jstr == NULL) {
         return; /* out of memory */
     }
     (*env)->SetObjectField(env, obj, fid, jstr);
 }
```

Uzyskanie sygnatur metod i pól

- javap -s -p Prompt

```
Compiled from Prompt.java
class Prompt extends java.lang.Object
    /* ACC_SUPER bit set */
{
    private native getLine
  (Ljava/lang/String;)Ljava/lang/String;
    public static main
  ([Ljava/lang/String;)V
    <init> ()V
    static <clinit> ()V
}
```

Typy sygnatur

Signature	Java Type
Z	boolean
B	byte
C	char
S	short
I	int
J	long
F	float
D	double
L fully-qualified-class ;	fully-qualified-class
[type	type[]
(arg-types) ret-type	method type

Dostęp do pól statycznych

```
class StaticFieldAccess {
    private static int si;

    private native void accessField();
    public static void main(String args[]) {
        StaticFieldAccess c = new StaticFieldAccess();
        StaticFieldAccess.si = 100;
        c.accessField();
        System.out.println("In Java:");
        System.out.println("  StaticFieldAccess.si = " +
   si);
    }
    static {
        System.loadLibrary("StaticFieldAccess");
    }
}
```

```
JNIEXPORT void JNICALL
 Java_StaticFieldAccess_accessField(JNIEnv *env, jobject
   obj)
 {
     jfieldID fid;   /* store the field ID */
     jint si;

     /* Get a reference to obj's class */
     jclass cls = (*env)->GetObjectClass(env, obj);

     printf("In C:\n");

     /* Look for the static field si in cls */
     fid = (*env)->GetStaticFieldID(env, cls, "si", "I");
     if (fid == NULL) {
         return; /* field not found */
     }
     /* Access the static field si */
     si = (*env)->GetStaticIntField(env, cls, fid);
     printf("  StaticFieldAccess.si = %d\n", si);
     (*env)->SetStaticIntField(env, cls, fid, 200);
 }
```

Wywoływanie metod

```
class InstanceMethodCall {
    private native void nativeMethod();
    private void callback() {
        System.out.println("In Java");
    }
    public static void main(String args[]) {
        InstanceMethodCall c = new
  InstanceMethodCall();
        c.nativeMethod();
    }
    static {
        System.loadLibrary("InstanceMethodCall");
    }
}
```

Wywołanie metod po stronie kodu natywnego

```
JNIEXPORT void JNICALL
 Java_InstanceMethodCall_nativeMethod(JNIEnv *env,
  jobject obj)
 {
     jclass cls = (*env)->GetObjectClass(env, obj);
     jmethodID mid =
          (*env)->GetMethodID(env, cls, "callback",
  "()V");
     if (mid == NULL) {
          return; /* method not found */
     }
     printf("In C\n");
     (*env)->CallVoidMethod(env, obj, mid);
 }
```

Wywołanie metody z interfejsu po stronie kodu natywnego

```
jobject thd = ...; /* a java.lang.Thread instance */
 jmethodID mid;
 jclass runnableIntf =
     (*env)->FindClass(env, "java/lang/Runnable");
 if (runnableIntf == NULL) {
     ... /* error handling */
 }
 mid = (*env)->GetMethodID(env, runnableIntf, "run",
   "()V");
 if (mid == NULL) {
     ... /* error handling */
 }
 (*env)->CallVoidMethod(env, thd, mid);
 ... /* check for possible exceptions */
```

Wywołanie metody statycznej

```
class StaticMethodCall {
    private native void nativeMethod();
    private static void callback() {
        System.out.println("In Java");
    }
    public static void main(String args[]) {
        StaticMethodCall c = new StaticMethodCall();
        c.nativeMethod();
    }
    static {
        System.loadLibrary("StaticMethodCall");
    }
}
```

Wywołanie metody statycznej po stronie kodu natywnego

```
JNIEXPORT void JNICALL
 Java_StaticMethodCall_nativeMethod(JNIEnv *env,
  jobject obj)
 {
     jclass cls = (*env)->GetObjectClass(env, obj);
     jmethodID mid =
          (*env)->GetStaticMethodID(env, cls,
  "callback", "()V");
     if (mid == NULL) {
          return;  /* method not found */
     }
     printf("In C\n");
     (*env)->CallStaticVoidMethod(env, cls, mid);
 }
```

Wywołanie konstruktora (1)

```
jstring MyNewString(JNIEnv *env, jchar *chars, jint len)
 {
     jclass stringClass;
     jmethodID cid;
     jcharArray elemArr;
     jstring result;

     stringClass = (*env)->FindClass(env, "java/lang/String");
     if (stringClass == NULL) { return NULL; /* exception thrown */
     }
 /* Get the method ID for the String(char[]) constructor */
     cid = (*env)->GetMethodID(env, stringClass, "<init>", "([C)V");
     if (cid == NULL) { return NULL; /* exception thrown */
     }
      /* Create a char[] that holds the string characters */
     elemArr = (*env)->NewCharArray(env, len);
     if (elemArr == NULL) { return NULL; /* exception thrown */
    }
     (*env)->SetCharArrayRegion(env, elemArr, 0, len, chars);

     /* Construct a java.lang.String object */
     result = (*env)->NewObject(env, stringClass, cid, elemArr);
      /* Free local references */
     (*env)->DeleteLocalRef(env, elemArr);
     (*env)->DeleteLocalRef(env, stringClass);
     return result;
 }
```

Wywołanie konstruktora (2)

```
result = (*env)->AllocObject(env, stringClass);
 if (result) {
     (*env)->CallNonvirtualVoidMethod(env, result,
   stringClass,
                                      cid, elemArr);
     /* we need to check for possible exceptions */
     if ((*env)->ExceptionCheck(env)) {
         (*env)->DeleteLocalRef(env, result);
         result = NULL;
     }
 }
```

Wyjątki

- Strukturalna obsługa wyjątków w C++

```
try
{
    throw "Powód wyjątku";
}
catch (char *exception)
{
    printf("Wyjątek z powodu: %s",exception);
}
```

Wyjątki w kodzie JAVA i kodzie natywnym

```
class CatchThrow {
    private native void doit()
        throws IllegalArgumentException;
    private void callback() throws NullPointerException {
        throw new NullPointerException("CatchThrow.callback");
    }
    public static void main(String args[]) {
        CatchThrow c = new CatchThrow();
        try {
            c.doit();
        } catch (Exception e) {
            System.out.println("In Java:\n\t" + e);
        }
    }
    static {
        System.loadLibrary("CatchThrow");
    }
}
```

```
JNIEXPORT void JNICALL
 Java_CatchThrow_doit(JNIEnv *env, jobject obj)
 {
     jthrowable exc;
     jclass cls = (*env)->GetObjectClass(env, obj);
     jmethodID mid =
         (*env)->GetMethodID(env, cls, "callback", "()V");
     if (mid == NULL) {
         return;
     }
     (*env)->CallVoidMethod(env, obj, mid);
     exc = (*env)->ExceptionOccurred(env);
     if (exc) {
         jclass newExcCls;
         (*env)->ExceptionDescribe(env);
         (*env)->ExceptionClear(env);
         newExcCls = (*env)->FindClass(env,
                             "java/lang/IllegalArgumentException");
         if (newExcCls == NULL) {
             /* Unable to find the exception class, give up. */
             return;
         }
         (*env)->ThrowNew(env, newExcCls, "thrown from C code");
     }
 }
```

Metody do obsługi wyjątków w JNI

- `throw()`
  - wyrzucenie wyjątku. Funkcja używana w natywnej metodzie do ponownego wyrzucenia wyjątku
- `throwNew()`
  - tworzy nowy obiekt wyjątku i go wyrzuca
- `exceptionDescribe()`
  - wypisuje ślad stosu i wyjątek
- `exceptionOccurred()`
  - pozwala stwierdzić czy wyrzucono wyjątek
- `exceptionClear()`
  - „czyści” obsługiwany wyjątek
- `fatalError()`
  - zgłosza błąd „fatalny” i nie zwraca niczego

Stworzenie wirtualnej maszyny w kodzie natywnym

```
public class Prog {
    public static void main(String[] args) {
        System.out.println("Hello World " + args[0]);
    }
}
```

```c
#include <jni.h>
#define PATH_SEPARATOR ';' /* define it to be ':' on Solaris */
#define USER_CLASSPATH "." /* where Prog.class is */

 main() {
    JNIEnv *env;  JavaVM *jvm; jint res;  jclass cls;
    jmethodID mid;   jstring jstr; jclass stringClass;   jobjectArray args;

#ifdef JNI_VERSION_1_2
     JavaVMInitArgs vm_args;   JavaVMOption options[1];
     options[0].optionString = "-Djava.class.path=" USER_CLASSPATH;
     vm_args.version = 0x00010002; vm_args.options = options;
     vm_args.nOptions = 1; vm_args.ignoreUnrecognized = JNI_TRUE;
     /* Create the Java VM */
     res = JNI_CreateJavaVM(&jvm, (void**)&env, &vm_args);
#endif
     if (res < 0) { fprintf(stderr, "Can't create Java VM\n"); exit(1); }
     cls = (*env)->FindClass(env, "Prog");
     if (cls == NULL) { goto destroy; }
      mid = (*env)->GetStaticMethodID(env, cls, "main", "([Ljava/lang/String;)V");
     if (mid == NULL) { goto destroy; }
     jstr = (*env)->NewStringUTF(env, " from C!");
     if (jstr == NULL) { goto destroy; }
     stringClass = (*env)->FindClass(env, "java/lang/String");
     args = (*env)->NewObjectArray(env, 1, stringClass, jstr);
     if (args == NULL) { goto destroy; }
     (*env)->CallStaticVoidMethod(env, cls, mid, args);
  destroy:
     if ((*env)->ExceptionOccurred(env)) {(*env)->ExceptionDescribe(env); }
     (*jvm)->DestroyJavaVM(jvm); }
```

Linkowanie z JAVA

- Solaris

  cc -I<jni.h dir> -L<libjava.so dir> -lthread -ljava invoke.c

```
void *JNU_FindCreateJavaVM(char *vmlibpath)
{
    void *libVM = dlopen(vmlibpath, RTLD_LAZY);
    if (libVM == NULL) {
        return NULL;
    }
    return dlsym(libVM, "JNI_CreateJavaVM");
}
```

- Windows

  cl -I<jni.h dir> -MD invoke.c -link <javai.lib dir>\javai.lib

```
void *JNU_FindCreateJavaVM(char *vmlibpath)
{
    HINSTANCE hVM = LoadLibrary(vmlibpath);
    if (hVM == NULL) {
        return NULL;
    }
    return GetProcAddress(hVM, "JNI_CreateJavaVM");
}
```

Threads

- Here are basics of how to make JNI threadsafe:
  - The JNIEnv pointer is valid only for the thread associated with it. Don’t pass it around.
  - Local references are only valid in the thread that created them.
  - You can convert a local reference to global reference to share it.
  - You have the same sorts of synchronised problems in JNI you do in regular Java, it is just you deal with them with MonitorEnter and MonitorExit. For those who like to line dangerously, it is also possible to use the native thread mechanisms.

JNI Methods Useful In Reflection

Method	Use
GetSuperclass	returns superclass of a class reference.
IsAssignableFrom	checks whether instances of one class can be used when instances of another are expected.
GetObjectClass	return the class of a given jobject.
IsInstanceOf	checks whether a jobject is an instance of a given class.
FromReflectedField	Convert a java.lang.reflect.Field to a field ID.
ToReflectedField	Convert a to a field ID to a java.lang.reflect. Field.
FromReflectedMethod	Convert java.lang.reflect.Method or a java.lang.reflect. Constructor to a method ID.
ToReflectedMethod	Convert method ID to a java.lang.reflect.Method or a java.lang.reflect. Constructor.