JAVA NATIVE INTERFACE

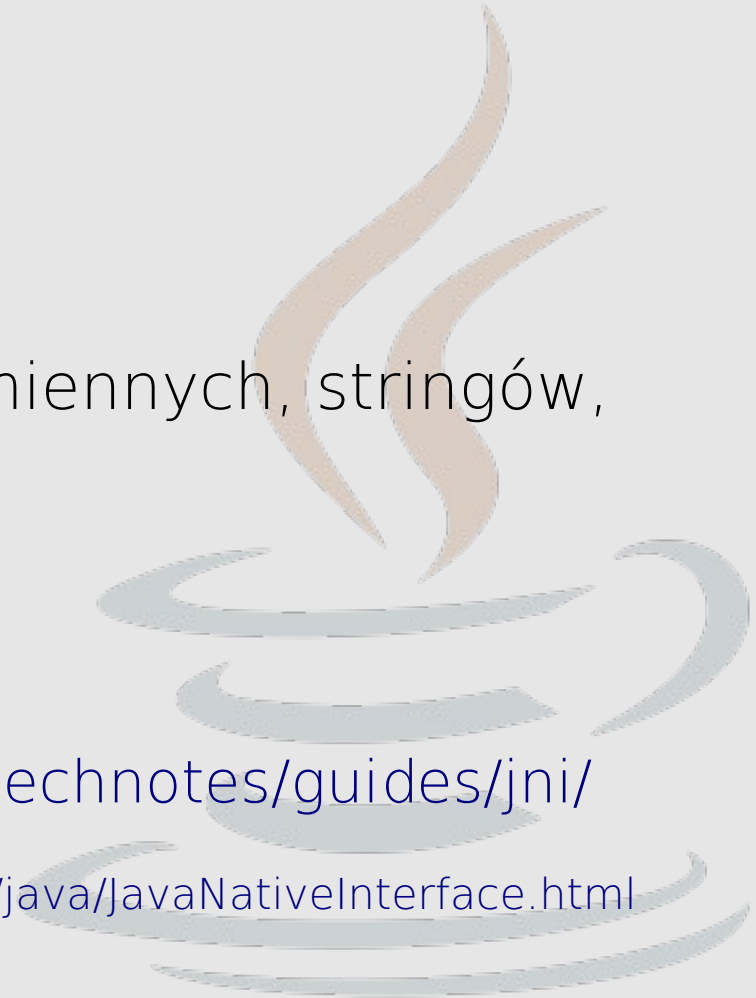
ZAGADNIENIA:

- wprowadzenie
- przykład HelloWorld
- podstawy JNI (przekazywanie zmiennych, stringów, obiektów)

MATERIAŁY:

http://docs.oracle.com/javase/7/docs/technotes/guides/jni/

http://www3.ntu.edu.sg/home/ehchua/programming/java/JavaNativeInterface.html



WPROWADZENIE

Java Native Interface umożliwia:

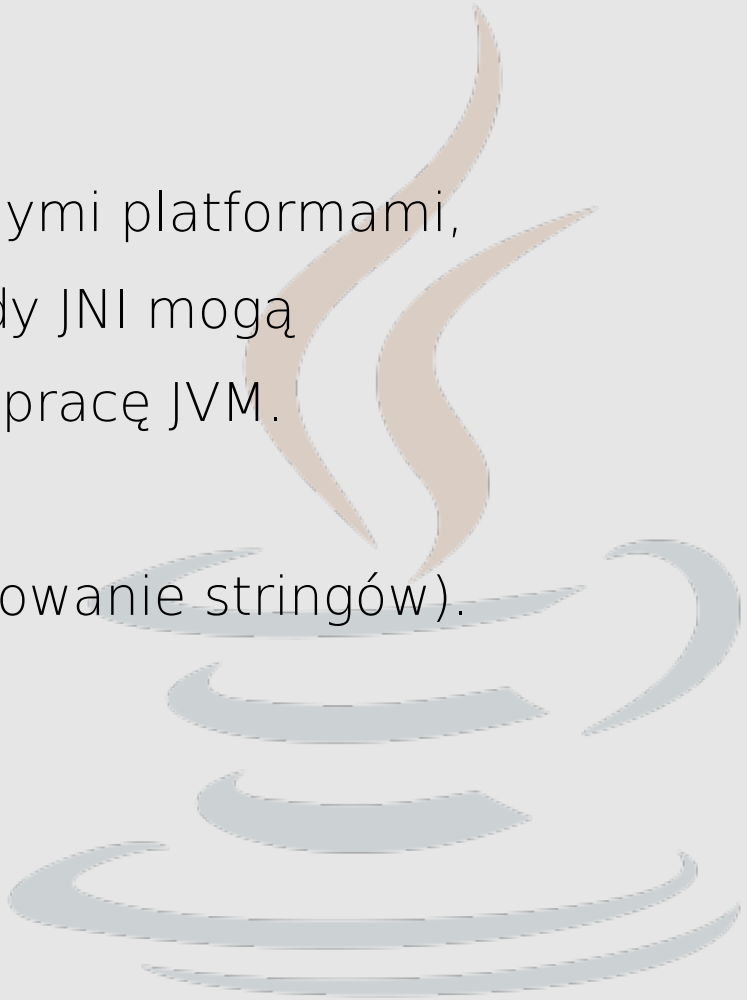
- korzystanie z bibliotek systemowych (napisanych np. w C/C++) wewnątrz programu w Javie.
- korzystanie z obiektów Javy wewnątrz programów w innych językach programowania (np. C/C++).

Najczęściej JNI wykorzystuje się do realizacji niskopoziomowej komunikacji IO (np. obsługa interfejsów USB, bluetooth), która nie może być zrealizowana z poziomu JVM, gdyż nie ma ona bezpośredniego dostępu do sprzętu.

WPROWADZENIE

Najważniejsze ograniczenia i wady JNI.

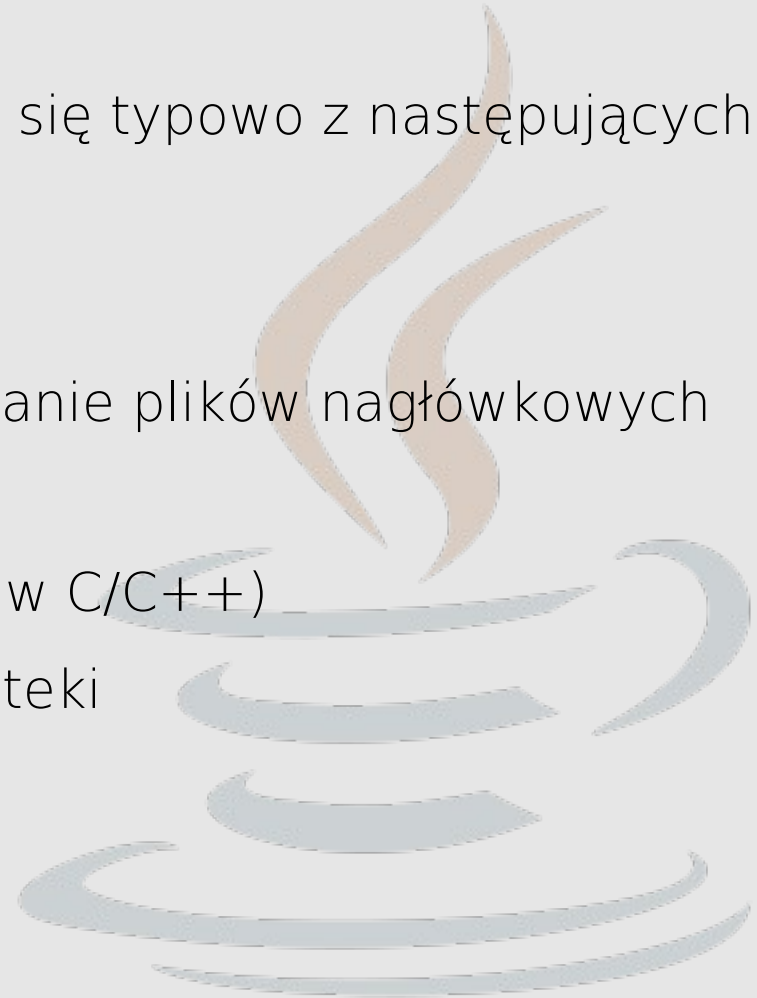
- brak przenośności kodu pomiędzy różnymi platformami,
- utrudnione debugowanie – drobne błędy JNI mogą "niedeterministycznie" destabilizować pracę JVM.
- brak garbage collectora,
- problemy z konwersją danych (np. kodowanie stringów).



HELLO WORLD

Stworzenie aplikacji używającej JNI składa się typowo z następujących kroków:

- napisanie programu w Javie
- skompilowanie programu I wygenerowanie plików nagłówkowych dla metod natywnych
- implementacja metod natywnych (np. w C/C++)
- kompilacja metod natywnych do biblioteki
- uruchomienie programu.



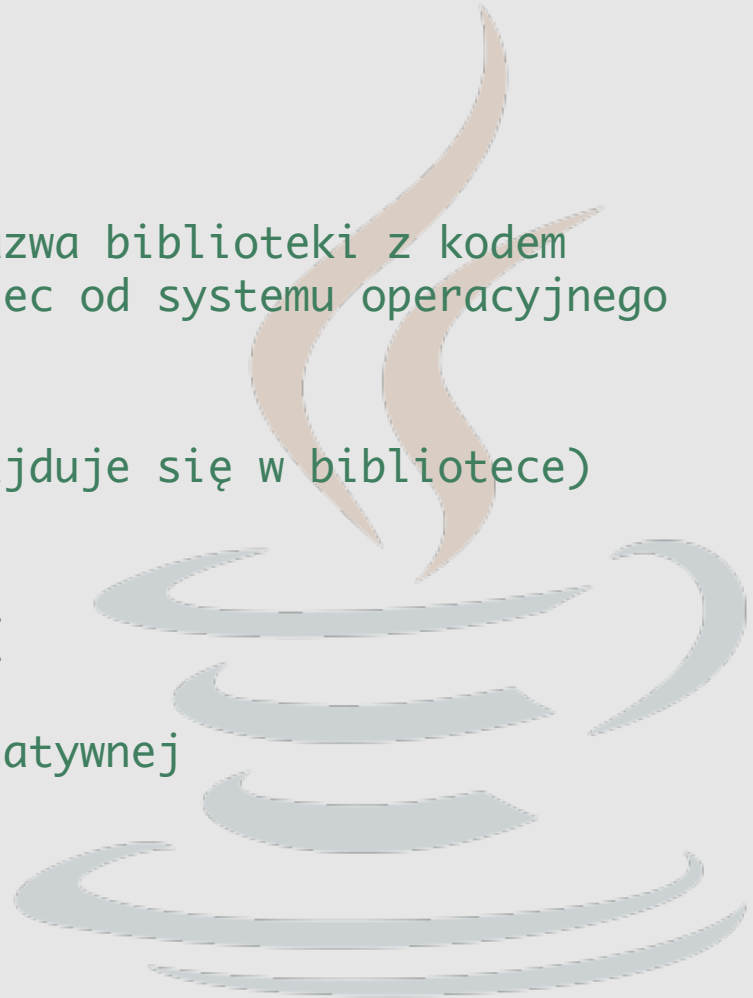
PROGRAM W JAVIE

```
public class JNIHello {

    static {
          System.loadLibrary("hello"); // nazwa biblioteki z kodem
                    // natywnym, moze zalezec od systemu operacyjnego
    }

    // deklaracja metody natywnej (ktora znajduje się w bibliotece)
    private native void sayHello();

    public static void main(String[] args) {
        JNIHello h = new JNIHello();
        h.sayHello();  // wywolanie metody natywnej
    }
}
```



KOMPILACJA

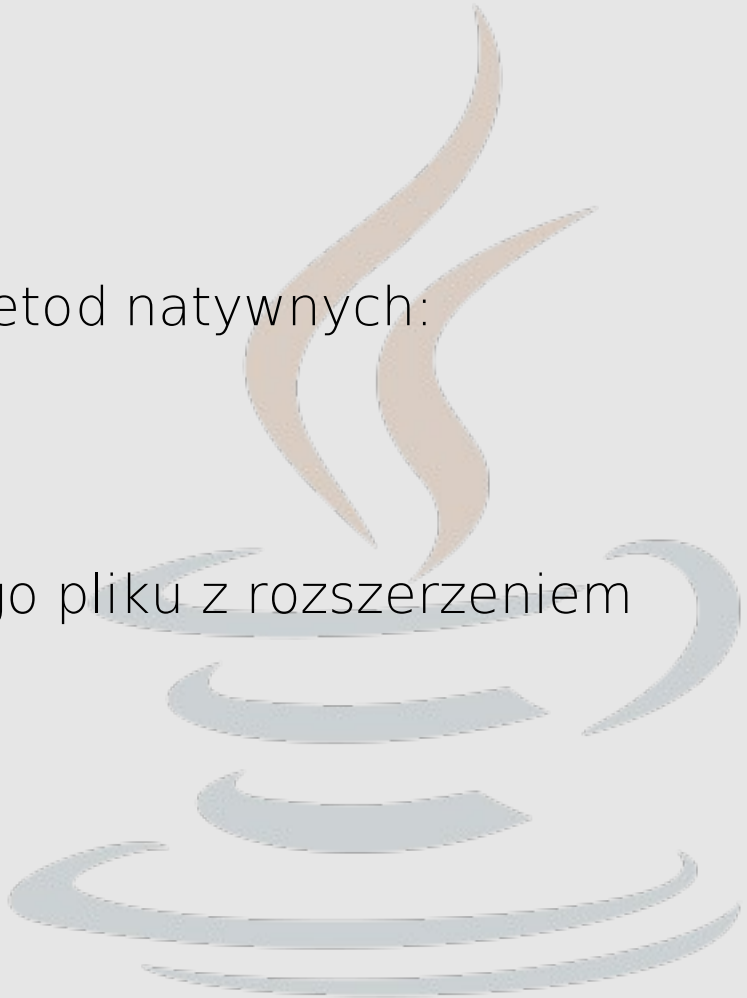
Kompilacja programu:

`javac JNIHello.java`

Generowanie plików nagłówkowych dla metod natywnych:

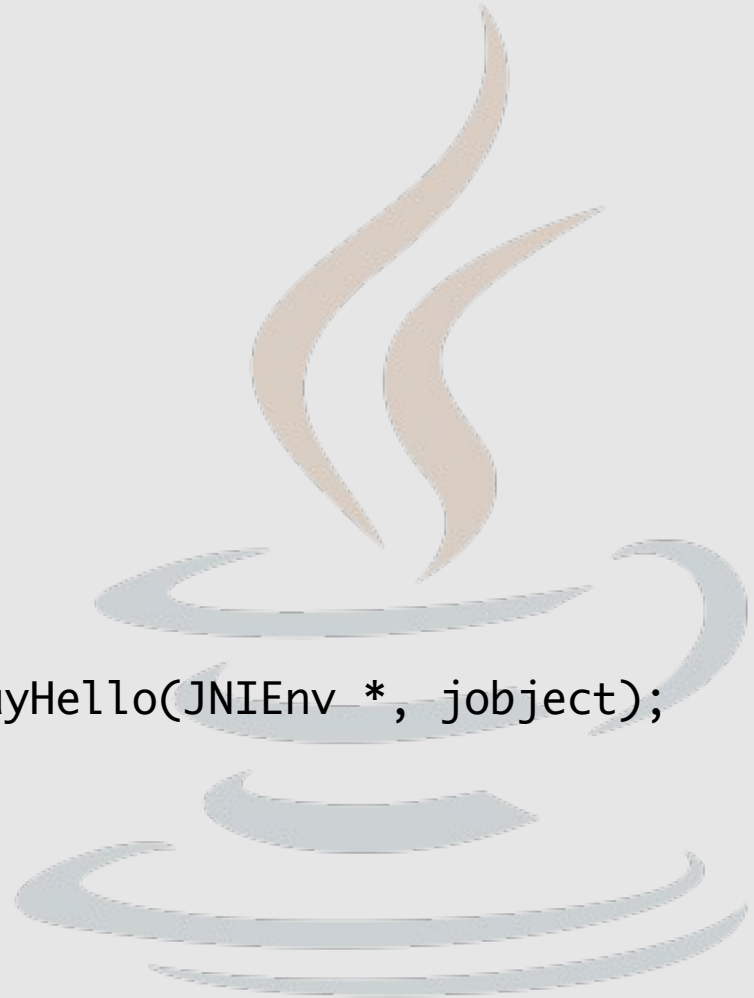
`javah JNIHello`

Do generowania używamy skompilowanego pliku z rozszerzeniem `class`.



PLIK NAGŁÓWKOWY

```
#include <jni.h>
/* Header for class JNIHello */
#ifndef _Included_JNIHello
    #define _Included_JNIHello
    #ifdef __cplusplus
        extern "C" {
    #endif
    /*
     * Class:     JNIHello
     * Method:    sayHello
     * Signature: ()V
     */
     JNIEXPORT void JNICALL Java_JNIHello_sayHello(JNIEnv *, jobject);
     #ifdef __cplusplus
        }
    #endif
#endif
```



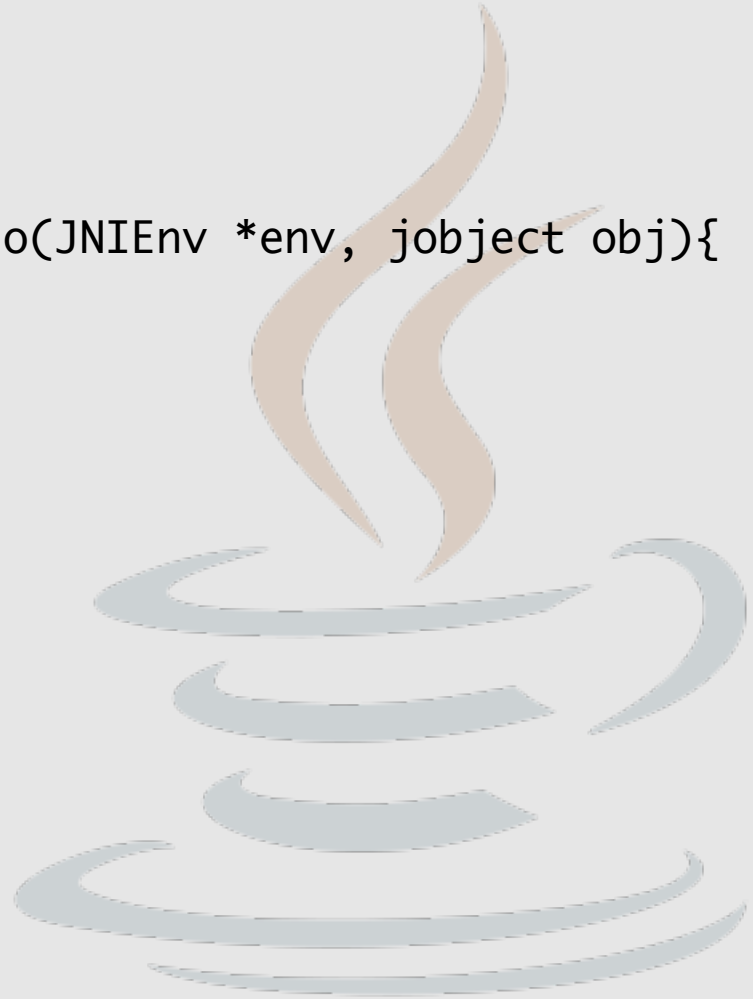
IMPLEMENTACJA

```
#include “JNIHello.h”
#include <stdio.h>
JNIEXPORT void JNICALL Java_JNIHello_sayHello(JNIEnv *env, jobject obj){

   printf("Hello World!\n");

   return;

}
```



KOMPILACJA BIBLIOTEKI

OSX:

```
gcc -I /System/Library/Frameworks/JavaVM.framework/Headers
 -c hello.c
gcc -I /System/Library/Frameworks/JavaVM.framework/Headers
 -shared -dynamiclib -o libhello.jnilib hello.o
```

Linux:

```
gcc -I /etc-java-config-2/current-system-vm/include
-I /etc-java-config-2/current-system-vm/include/linux -fPIC
-shared -o libhello.so hello.c
```

URUCHOMIENIE

Kompilacja biblioteki

OSX:

```
java JNIHello
```

Linux:

```
java -Djava.library.path=. JNIHello
```

PODSTAWY JNI

JNI definiuje następujące typy natywne:

- jint, jbyte, jshort, jlong, jfloat, jdouble, jchar, jboolean.
- jobject, jclass, jstring, jthrowable
- jarray (jintArray, jbyteArray, jshortArray, jlongArray, jfloatArray, jdoubleArray, jcharArray and jbooleanArray oraz jobjectArray).

Typy natywne mogą być przekazywane jako argumenty wywołania oraz mogą być zwracane. Jednak funkcje natywne operują na swoich natywnych typach (np int[], char*). Dlatego wymagana jest konwersja.

TYPY PRYMITYWNE

```
private native double add(double d1, double d2);

public static void main(String[] args){
    JNIExamples ex = new JNIExamples();
    System.out.println(ex.add(12.5, 5.3));
}
```

```
JNIEXPORT jdouble JNICALL Java_JNIExamples_add(JNIEnv *env, jobject obj,
jdouble d1, jdouble d2){
    printf("Otrzymade argumenty %f; %f \n",d1, d2);
    jdouble res = d1 + d2;
    return res;
}
```

STRINGI

```
private native String sayHello(String msg);
```

```
JNIEXPORT jstring JNICALL Java_JNIExamples_sayHello(JNIEnv *env, jobject
obj, jstring jin){
    // konwersja: (jstring) --> (char*)
    const char *in = (*env)->GetStringUTFChars(env, jin, NULL);
    if (NULL == in) return NULL;

    // dzialanie:
    printf("odebralem %s\n", in);
    // zwolnienie zasobow
    (*env)->ReleaseStringUTFChars(env, jin, in);  // release resources

    // przygotowanie odpowiedzi
    char out[] = "from C";
    // konwersja
    return (*env)->NewStringUTF(env, out);
}
```

STRINGI (UTF-8)

```
// UTF-8 String --> null-terminated char-array
const char * GetStringUTFChars(JNIEnv *env, jstring string, jboolean
*isCopy);

// Informije JVM ze kod natywny nie bedzie potrzebowal utf
void ReleaseStringUTFChars(JNIEnv *env, jstring string, const char *utf);

// Tworzy java.lang.String ze znakow
jstring NewStringUTF(JNIEnv *env, const char *bytes);

// zwraca długosc stringu
jsize GetStringUTFLength(JNIEnv *env, jstring string);

// w buforze zostanie zapisany wskazany fragment str
void GetStringUTFRegion(JNIEnv *env, jstring str, jsize start, jsize
length, char *buf);
```

STRINGI (UNICODE)

```
// Unicode String --> jchar*
const jchar * GetStringChars(JNIEnv *env, jstring string, jboolean
*isCopy);


void ReleaseStringChars(JNIEnv *env, jstring string, const jchar *chars);


jstring NewString(JNIEnv *env, const jchar *unicodeChars, jsize length);


jsize GetStringLength(JNIEnv *env, jstring string);


void GetStringRegion(JNIEnv *env, jstring str, jsize start, jsize length,
jchar *buf);
```

TABLICE

```
private native int[] swap(int[] numbers);
...
   int[] in = {1,2};
   int[] out = ex.swap(in);
   for(int i: out)
     System.out.println(i);
...

JNIEXPORT jintArray JNICALL Java_JNIExamples_swap(JNIEnv *env,
jobject obj, jintArray ia){
    jint *inArray = (*env)->GetIntArrayElements(env, ia, NULL);
    if (NULL == inArray) return NULL;
    jint outArray[] = {inArray[1], inArray[0]}; // dzialanie
    (*env)->ReleaseIntArrayElements(env, ia, inArray, 0); // zwolnienie
    jintArray out = (*env)->NewIntArray(env, 2);  // wynik
    if (NULL == out) return NULL;
    (*env)->SetIntArrayRegion(env, out, 0 , 2, outArray);  // kopiowanie
    return out;
}
```

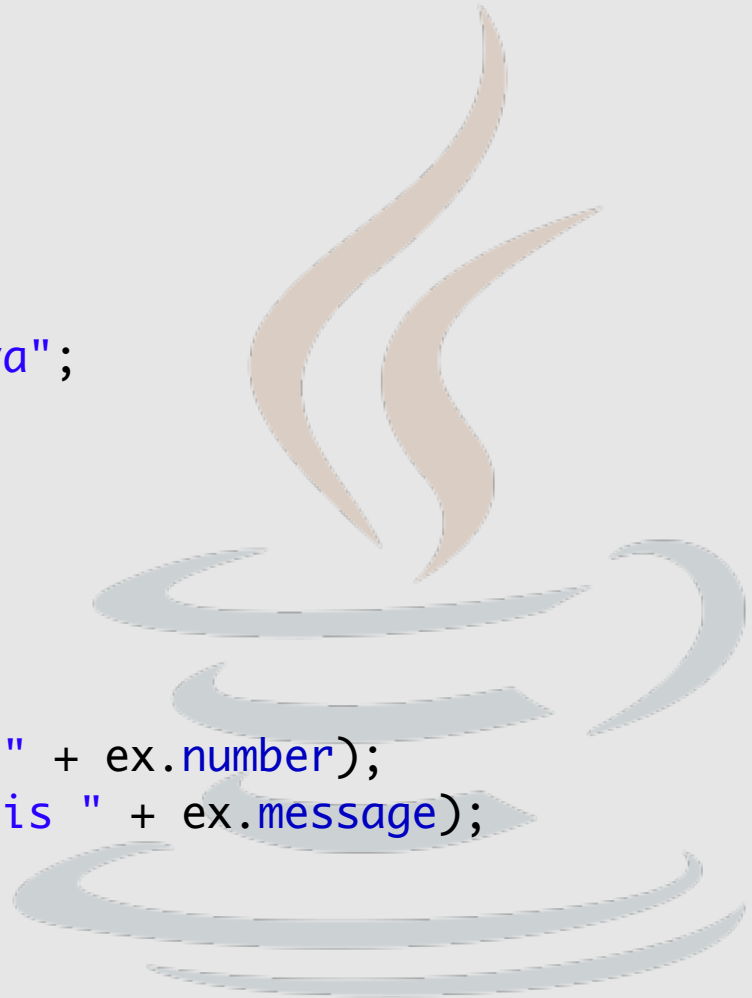
DOSTĘP DO OBIEKTÓW

```
public class JNIExamples {
    static {
        System.loadLibrary("examples");
    }

    private int number = 88;
    private String message = "Hello from Java";

    private native void modifyVariables();

    public static void main(String[] args) {
        JNIExamples ex = new JNIExamples();
        ex.modifyVariables();
        System.out.println("In Java, int is " + ex.number);
        System.out.println("In Java, String is " + ex.message);
    }
}
```



DOSTĘP DO OBIEKTÓW

```
JNIEXPORT void JNICALL Java_JNIExamples_modifyVariables(JNIEnv *env,
jobject thisObj) {

    // referencja do obiektu
    jclass thisClass = (*env)->GetObjectClass(env, thisObj);

    // pobranie pola int
    jfieldID fidNumber = (*env)->GetFieldID(env, thisClass,"number","I");
    if (NULL == fidNumber) return;

    // pobranie wartosci
    jint number = (*env)->GetIntField(env, thisObj, fidNumber);
    printf("In C, the int is %d\n", number);

    // zmiana wartosci
    number = 99;
    (*env)->SetIntField(env, thisObj, fidNumber, number);
```

DOSTĘP DO OBIEKTÓW

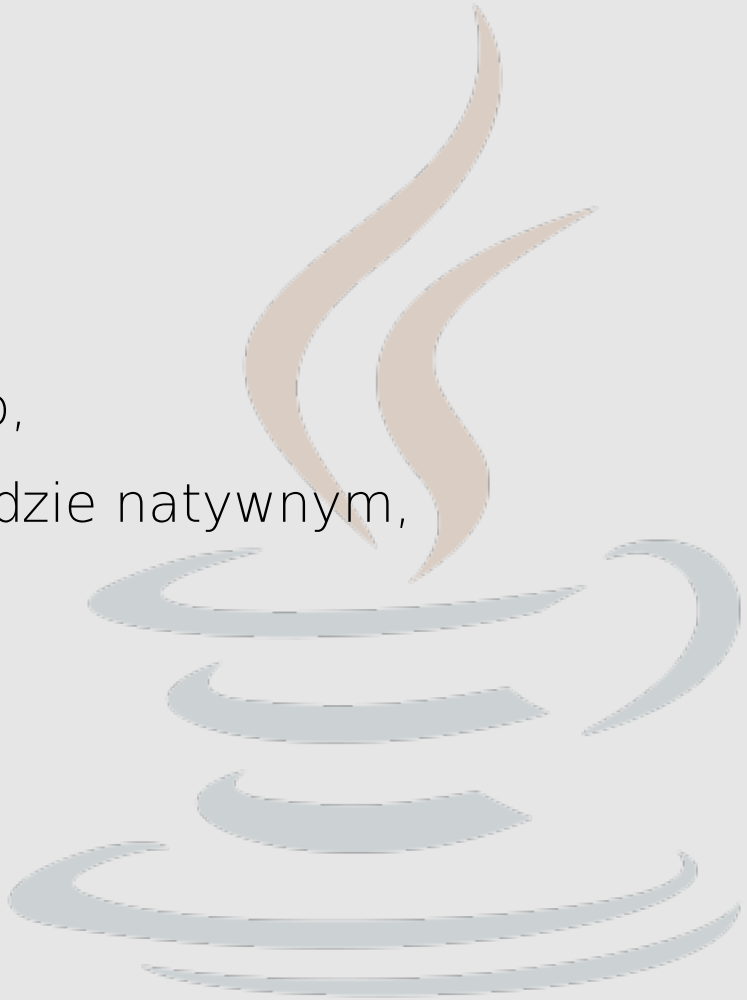
```
    // pobranie pola String
    jfieldID fidMessage = (*env)->GetFieldID(env, thisClass, "message",
                                                    "Ljava/lang/String;");
    if (NULL == fidMessage) return;

    // pobranie zawartosci
    jstring message = (*env)->GetObjectField(env, thisObj, fidMessage);

    // konwersja
    const char *cStr = (*env)->GetStringUTFChars(env, message, NULL);
    if (NULL == cStr) return;
    printf("In C, the string is %s\n", cStr);
    (*env)->ReleaseStringUTFChars(env, message, cStr); // porzadki
    // nowa wartosc
    message = (*env)->NewStringUTF(env, "Hello from C");
    if (NULL == message) return;
    // modyfikacja - podstawienie nowej wartosci
    (*env)->SetObjectField(env, thisObj, fidMessage, message);
}
```

INNE MOŻLIWOŚCI

- zmiana zmiennych statycznych,
- wywoływanie metod z kodu natywnego,
- tworzenie obiektów Javy (i tablic) w kodzie natywnym,
- zarządzanie referencjami.



DZIĘKUJĘ ZA UWAGĘ