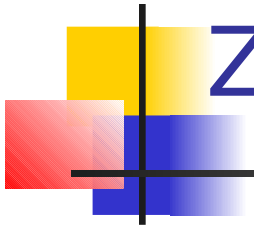
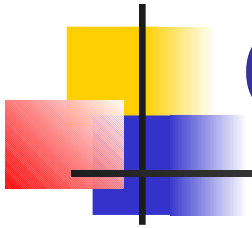


Zaawansowane technologie Javy

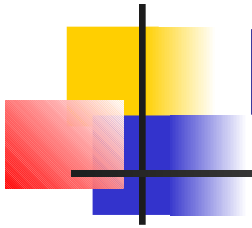
Wykład 11 (24 kwietnia 2012)

JNI – trochę C (a może Prologa?) w Javie



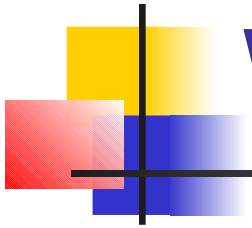
Co to jest JNI?

- Mechanizm do uruchamia w JVM kodu pisanego w innych językach
- Głównie C, C++, Assembler
- Ale istnieją zewnętrzne biblioteki umożliwiające obsługę innych języków



Kiedy używać JNI?

- Gdy potrzebne są rzeczy niedostępne w Javie, zależne od sprzętu, systemu itp. (Np. globalne skróty klawiaturowe)
- Gdy korzystamy z istniejącego już kodu/systemu/biblioteki i jego ponowna implementacja w Javie to zbyt duży koszt
- Gdy potrzebujemy kodu napisanego jak najbardziej efektywnie (np. w assemblerze)



Wady JNI

- Błąd w kodzie natywnym bywa trudny do wykrycia i bywa niebezpieczny dla części pisanej w Javie
- Tracimy silną kontrolę typów Javy np. na rzecz luźniejszej w C (kolejna okazja dla błędów)
- Z tych powodów najlepiej jest projektować kod tak, żeby metody natywne były skupione w poszczególnych klasach, których jest niewiele.
- Pewien narzut czasowy związany z wywołaniem metody w innym języku.
- Utrata wieloplatformowości. Sami musimy troszczyć o odpowiednie wersje rodzimych bibliotek dla różnych systemów.

Deklaracja metod natywnych

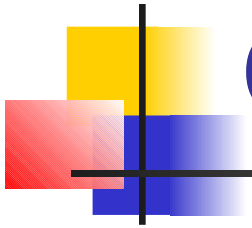
- Słowo kluczowe `native`.
- Przykładowa klasa „`HelloNative.java`”:

```
class HelloNative{
   public static native void greeting();
}
```

- Deklarujemy jak metodę abstrakcyjną
- Treść metody w bibliotece zewnętrznej

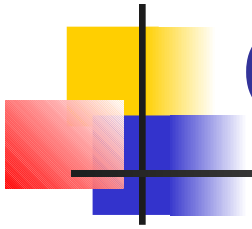
nagłówkowy

```
/* Tu była linijka ale ją usunęliśmy */
#include <jni.h>
/* Header for class HelloNative */

#ifndef _Included_HelloNative
#define _Included_HelloNative
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     HelloNative
 * Method:    greeting
 * Signature: ()V
 */
JNIEXPORT void JNICALL Java_HelloNative_greeting
  (JNIEnv *, jclass);

#ifdef __cplusplus
}
#endif
#endif
```



Oto wycięta linijka:

- `/* DO NOT EDIT THIS FILE – it is machine generated */`
- Żeby wygenerować ten plik:
  - `javac HelloNative.java`
  - `javah HelloNative`



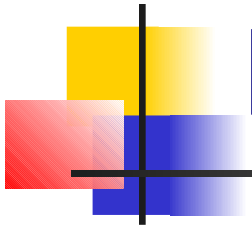
Char, String – jchar, jstring

- Java korzysta z kodowania UTF-16
- C korzysta z UTF-8
- jchar to kod UTF-16
- Otrzymujemy zestaw metod:
  - jstring NewStringUTF(JNIEnv*, const char[])
  - Jsize GetStringUTFLength(JNIEnv*, jstring)
  - … w książce

Wywoływanie metod JNIEnv

- Pierwsze parametry metody natywnej to zmienna środowiskowa `JNIEnv*` oraz struktura typu `jclass` lub `jobject`.
  - `JNIEnv*` wskazuje na tablicę wskaźników na funkcje

```
jstr=(*env)->NewStringUTF(env,
„Greetings!”); // w C


jstr=env->NewStringUTF(env,
„Greetings!”); // w C++
```



HelloNative.c

```
#include "HelloNative.h"
#include <stdio.h>

JNIEXPORT void JNICALL
Java_HelloNative_greeting(JNIEnv* env, jclass
cl){
    printf("Hello Native World!\n");
}
```

Kompilacja i uruchomienie

```
gcc -fPIC -I <katalog z JDK>/include -I
<katalog z JDK>/include/linux -shared -o
libHelloNative.so HelloNative.c
```

- W przypadku innych systemów lub innych kompilatorów prosimy o sięgnięcie do dokumentacji

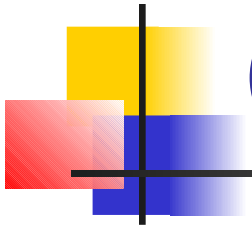
Ładowanie biblioteki

```
class HelloNative{
   public static native void greeting();

   static{
      System.loadLibrary(„HelloNative”)
   }
}
```

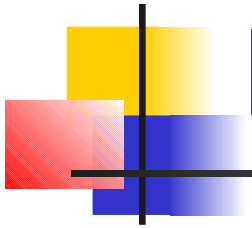
- Maszyna wirtualna musi znać ścieżkę dostępu do biblioteki, np:

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

Typy w kodzie natywnym (C/C++)

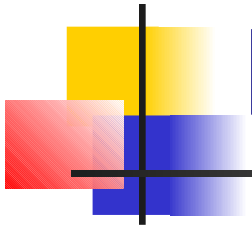


Java	C	Bajty
boolean	jboolean	1
byte	jbyte	1
char	jchar	2
short	jshort	2
int	jint	4
long	jlong	8
float	jfloat	4
double	jdouble	8



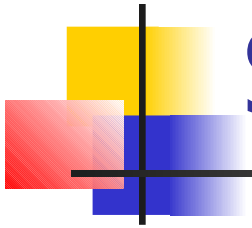
Dostęp do pól obiektu

```
jclass class_Employee =
   (*env)->GetObjectClass(env,this_obj);
jfieldID id_salary =
   (*env)->getFieldID(env, class_Employee, „salary”,”D”);
jdouble salary=
   (*env)->GetDoubleField(env, this_obj, id_salary);
salary*=1+byPercent/100;
(*env)->setDoubleField(env,this_obj,id_salary,salary);
```



Dostęp do pól statycznych

```
jclass class_System=
   (*env)->FindClass(env, „java/lang/System”);
jfieldID id_out=
   (*env)->GetStaticFieldID(env, class_System,
   „out”, „Ljava/io/PrintStream; ”);
jobject obj_out =
(*env)->GetStaticObjectField(env,class_System,
id_out);
```



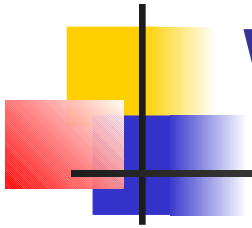
Sygnatury

- B byte
- C char
- D double
- F float
- I int
- J long
- Lclassname typ będący klasą
- S short
- V void
- Z boolean
- [tablica

- A tak naprawdę, to:
  `javap -s -private HelloNative`

Wywoływanie metod obiektów

```
class_PrintWriter =
   (*env)->GetObjectClass(env,out);
id_print =
(*env)->GetMethodID(env, class_PrintWriter,
„print”, „Ljava/lang/String;)V”);
(*env)->CallVoidMethod(env,out,id_print,str);
```

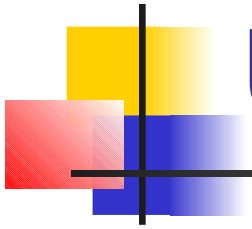


Wyjątkowy program

```
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");
    }
}
```

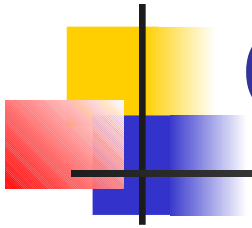
Wyjątkowy program 2

```
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");
    }
}
```



Uwaga!

- Obecnie wyrzucanie wyjątków przez metody macierzyste tworzone w języku C++ nie jest zaimplementowane. Dlatego w kodzie macierzystym rzucamy wyjątki metodami `Throw` i `ThrowNew` bez wykorzystywania mechanizmów C++!



Coś i bibliografia

- LibreOffice Impress robi lepsze wcięcia niż DevC++, ale wie lepiej co człowiek ma na myśli.
- http://docs.oracle.com/javase/7/docs/technotes/guides/jni/spec/jniTOC.html
- http://java.sun.com/docs/books/jni/html/titlepage.html
- Cay Horstmann, Gary Cornell: Core Java. Techniki zaawansowane. Wydanie 8. Wydawnictwo HELION, Gliwice 2009.