
Cours Java Native Interface

1999, Cnam Paris

jean-michel Douin, douin@cnam.fr

Version du 28 Mars 2002

http://lmi92.cnam.fr:8080/tp_cdi/{douin/}

Notes de cours jni : de Java vers C/C++ et de C/C++ vers Java

Sommaire

- **De Java vers C/C++**

- **De C/C++ vers Java**

lecture/écriture de données d'instance et de classes

invocation de méthodes d'instance et de classes

création d'objet

création de tableaux et de String

Levée et filtrage d'exceptions

utilisation des moniteurs (de hoare)

Entrées/sorties Série

création de machine(s) Java

API JNI // Architecture de la machine virtuelle

Ces notes sont accessibles : http://lmi92.cnam.fr/tp_cdi/java_jni.pdf

Bibliographie utilisée

Le tutorial JNI de Beth Stearns

- <http://java.sun.com/docs/books/tutorial/index.html>
- surtout : <http://java.sun.com/docs/books/tutorial/native1.1/index.html>
- jni specification : <http://java.sun.com/products/jdk/1.1/download-pdf-ps.html>

Un « essentiel » ouvrage sur le sujet

- Essential JNI: Java Native Interface, rob Gordon, Prentice hall.1998
ISBN 0-13-679895-0
- le site de l'éditeur <http://www.phptr.com/>

les instructions et l'architecture de la machine virtuelle Java reflètent les fonctionnalités de cette API

- The VM specification. <http://java.sun.com/docs/books/vmspec/html>

JNI Pourquoi ?

- **Applications existantes dans un environnement Java, avec ou sans les sources...**
- **Programmation d'un périphérique, logiciel de base, Entrées/Sorties, Cartes d'acquisition, de commandes**
(Adressage physique, Accès au matériel, aux pilotes de la carte, interruptions...)
- **Développement en C/C++, tout en bénéficiant de l'environnement Java IHM en Java, application en C, Applet, accès à l'internet**
- **Code Natif pour de meilleures performances en temps d'exécution**
- **Portabilité est annulée,**
dépendant de la plate-forme, moindres robustesse et sécurité

JNI Présentation

- **Deux aspects**

 - de Java vers C/C++

 - de C/C++ vers Java

 - en venant de Java

 - ou depuis une application ordinaire

- **L 'API JNI offre l 'accès à la machine virtuelle**

 - accès aux variables d 'instance, appel de méthodes, chargement d 'une classe, création d 'instances...

 - Mécanisme de bas-niveau...

 - Exceptions,

 - Threads....

De Java vers C(jc)

- **1) usage du mot clé native**
- **2) génération par les outils de SUN de l'interface « .h »** *>javah -jni Exemple*
- **3) génération de la DLL (Win32,.dll), ou de la SOL(solaris,.so)**
- **4) exécution**
 - nécessite l'accès à \$JDK_HOME\include
 - et en fonction de la plate-forme \$JDK_HOME\include\win32 (ou solaris)

jc) 1) Le source Java

- `public class JavaVersC {`
- `public native void bonjour(); // (1)`
- `public static void main(String args[]) {`
- `new JavaVersC().bonjour();`
- `}`
- `static{`
- `System.loadLibrary("JavaVersC"); // (2)`
- `}`
- `}`

- **>*javac JavaVersC.java***

- **(1) emploi de native**

- **(2) chargement de la librairie (DLL/sol) dans laquelle sera implémentée le code C de bonjour**

jc) Le source C : interface

- **2) génération de l'interface (.h) par javah**
- **>javah -jni JavaVersC**

```
• /* DO NOT EDIT THIS FILE - it is machine generated */
• #include <jni.h>
• /* Header for class JavaVersC */
• #ifndef _Included_JavaVersC
• #define _Included_JavaVersC
• #ifdef __cplusplus
• extern "C" {
• #endif
• /* Class:      JavaVersC
• * Method:     bonjour
• * Signature:  ()V
• */
• JNIEXPORT void JNICALL Java_JavaVersC_bonjour(JNIEnv *, jobject);
• #ifdef __cplusplus
• }
• #endif
• #endif
```


jc) le source C : Implémentation

- `#include <stdio.h>`
- **`#include "JavaVersC.h"`**
- `JNIEXPORT void JNICALL Java_JavaVersC_bonjour (JNIEnv *env, jobject j){`
- `printf("Java_JavaVersC_bonjour");`
- `}`

- **3) Génération de la DLL, (JavaVersC.dll)**
Avec visual c++
`cl -lc:\jdk\include -lc:\jdk\include\win32 -LD JavaVersC.c -FeJavaVersC.dll`

- **4) Exécution par**
`>java JavaVersC`

jc) JNIENV et jobject

- **JNIEnv *env**

Il s'agit de l'environnement de la machine Java associé au « Thread » courant, (le Thread ayant engendré l'appel de la méthode native `bonjour`)

- **jobject j**

Il s'agit de l'objet receveur du message `bonjour()`, ici l'instance créée dans la méthode `main`

- **En résumé**

A chaque appel d'une méthode native sont transmis par la machine Java un environnement

l'objet receveur ou la classe si c'est une méthode classe et éventuellement les paramètres

De C vers Java(cj)

- depuis une DLL/SO engendrée par javah
emploi du mot clé native

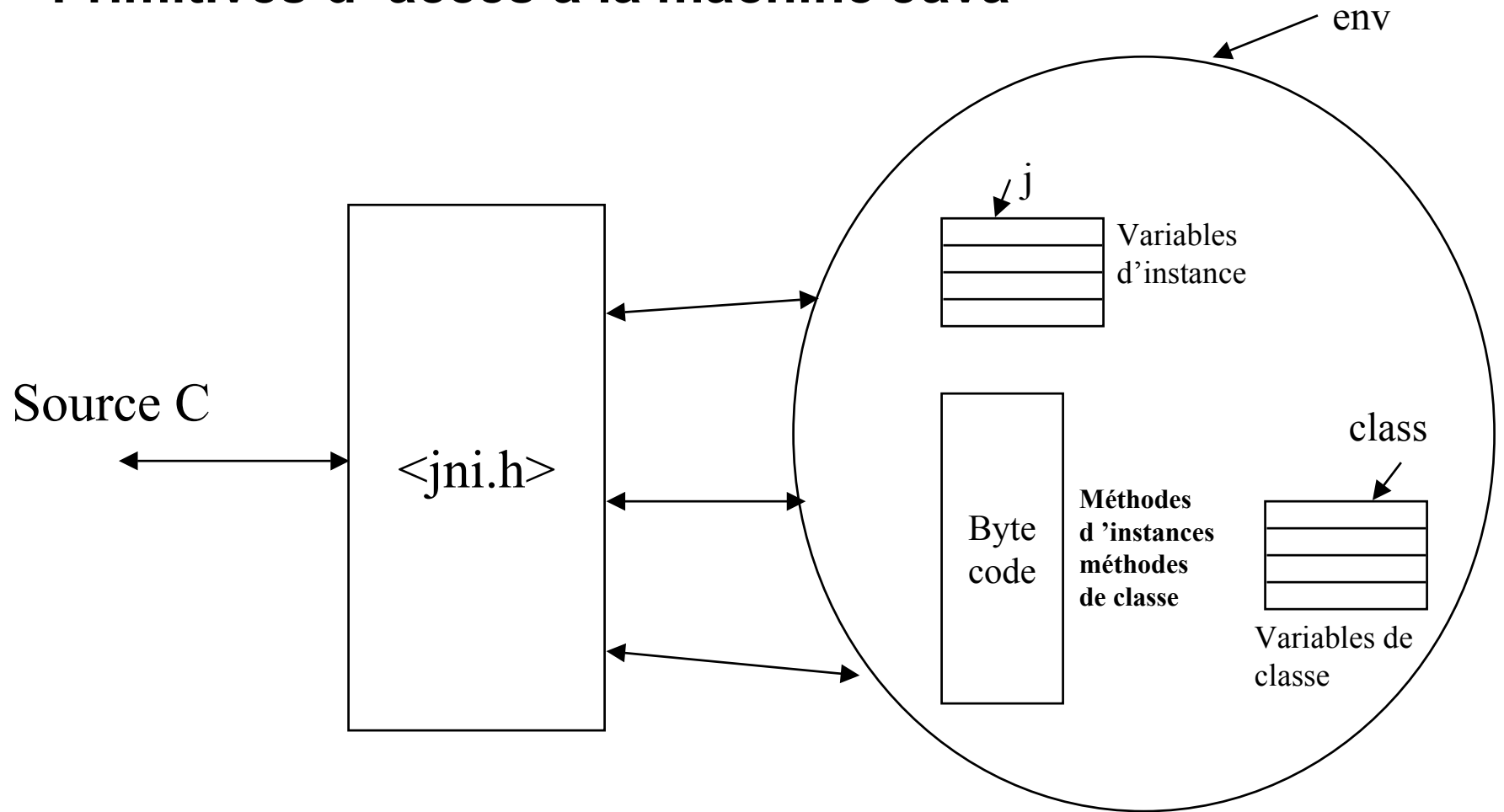
```
java ----native----->C/C++  
                        |  
java <----API_JNI----- C/C++
```

- ou depuis une application C/C++ ordinaire

```
java <----API_JNI----- C/C++
```

cj) De C vers Java

- Primitives d'accès à la machine Java



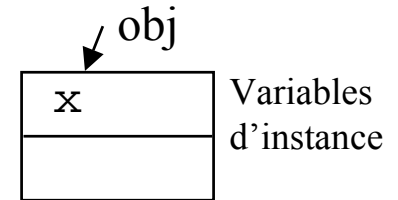
Fonctions de JNI

- **Versions**
- **Opérations sur les classes**
- **Exceptions**
- **Références locales et globales**
- **Opérations sur les objets**
- **Accès aux champs des objets**
- **Appels de méthodes d'instances**
- **Accès aux champs statiques**
- **Appels de méthodes de classes**
- **Opérations sur les instances de type String**
- **Opérations sur les tableaux**
- **Accès aux moniteurs**
- **Interface de la JVM**

cj) Accès aux Variables d'instance

- **GetFieldID, Get<type>Field, Set<type>Field**

```
public class Exemple{
    private int x;
    public native void setX(int val);
}
```



- **En C :**
- **JNIEXPORT void JNICALL Java_Exemple_setX (JNIEnv *env, jobject obj, jint val){**
 jclass classe = (*env)->GetObjectClass(env,obj);
 jfieldID fid = (*env)->GetFieldID(env,classe,"x","I");
 (*env)->SetIntField(env,obj,fid,val) ;
 }

instructions JVM : getfield, putfield

Exemple.java au complet

```
public class Exemple{
private int x;
public native void setX(int val);

    static{
        System.loadLibrary("Exemple");
    }

    public static void main(String args[]) {
        Exemple e = new Exemple();
        e.setX(33);
        System.out.println(" dites " + e.x);
    }
}
```

Exemple.c au complet

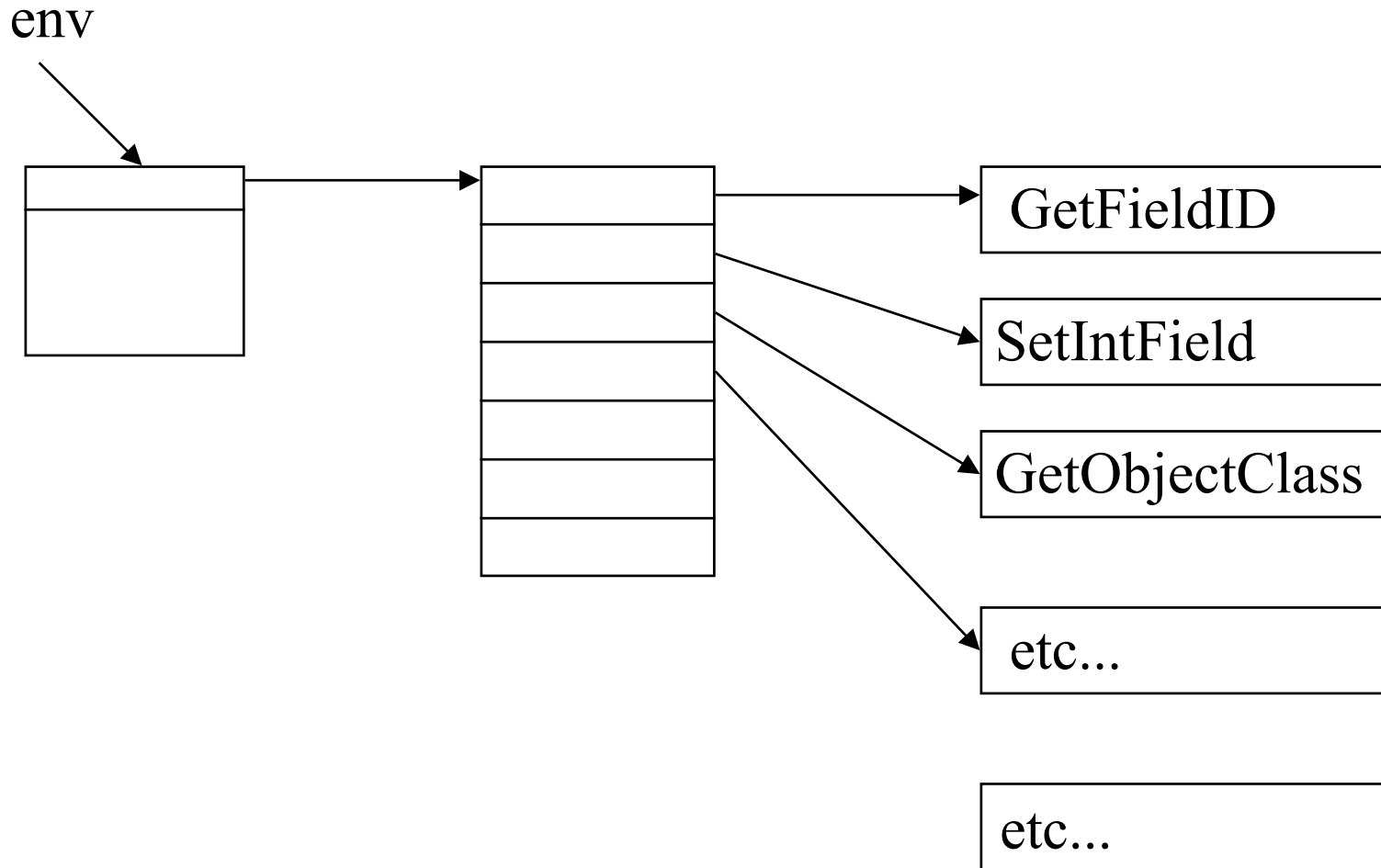
```
#include "Exemple.h"
```

```
JNIEXPORT void JNICALL Java_Exemple_setX(JNIEnv* env,  
    jobject obj, jint val){  
    jclass cl = (*env)->GetObjectClass(env,obj);  
    jfieldID fid = (*env)->GetFieldID(env,cl,"x","I");  
    (*env)->SetIntField(env,obj,fid,val) ;  
}
```


Les commandes Win32 au complet

- `javac -classpath . Exemple.java`
- `javah -jni -classpath . Exemple`
- `cl /Ic:\jdk\include /Ic:\jdk\include\win32 /LD Exemple.c /FeExemple.dll`
- `java -cp . Exemple`

Retour sur JNIENV *env



en C : `(*env)->GetFieldID(env,...)`

en C++ : `env->GetFieldID(...)`

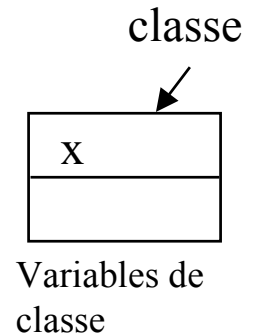
JNIENV *env

- **Inspiré du JRI de Netscape**
- **Chaque fonction est accessible par un déplacement fixe dans une table,**
- **Une table peut donc être substituée par une autre sans avoir à recompiler,**
- **L'implantation des fonctions de cette table est à la charge du fournisseur de la JVM,**
- **Le code C/C++ d'accès à la JVM devient portable**
- ***Analogue à l'implémentation de la table des méthodes virtuelles d'une instance***

cj) Accès aux Variables de classes

- **GetStaticFieldID, GetStatic<type>Field, SetStatic<type>Field**

```
• public class Exemple{  
    private static int x;  
    public native setStaticX(int val);  
}
```

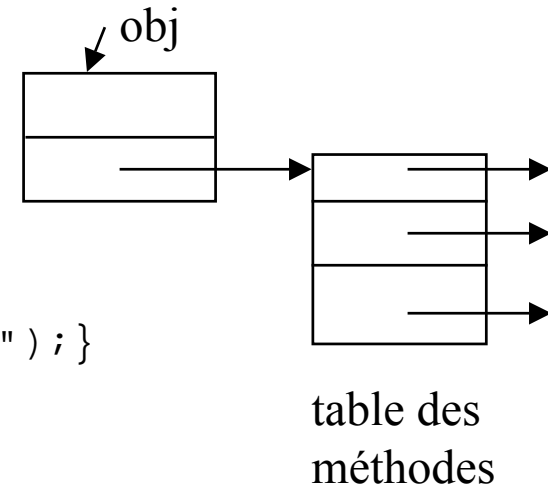


```
• JNIEXPORT void JNICALL Java_Exemple_setStaticX  
    (JNIEnv *env, jobject obj, jint val){  
    jclass classe = (*env)->GetObjectClass(env,obj);  
    jfieldID fid = (*env)->GetStaticFieldID(env,classe,"x","I");  
    (*env)->SetStaticIntField(env,classe,fid,val) ;  
}
```

instructions JVM : getstatic, putstatic

cj) Appels de méthodes d'instance

- **Call<type>Method**



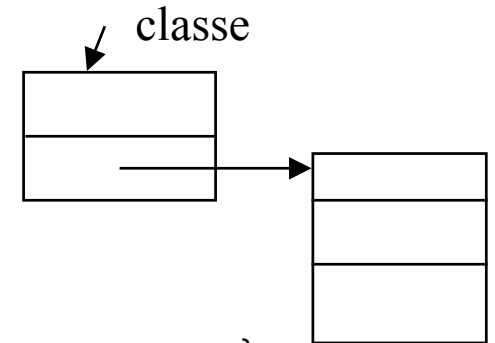
- ```
public class Exemple{
 public void p(){System.out.println("appel de p ");}
 public native callP(int val);
}
```

- ```
JNIEXPORT void JNICALL Java_Exemple_callP
                (JNIEnv *env, jobject obj){
    jclass classe = (*env)->GetObjectClass(env,obj);
    jMethodID mid = (*env)->GetMethodID(env,classe,"p","()V");
    (*env)->CallVoidMethod(env,obj,mid) ;
}
```

instruction JVM : invokevirtual

cj) Appels de méthodes de classe

- **CallStatic<type>Method**



- ```
public class Exemple{
 public static void p(){System.out.println("appel de p ");}
 public native callP(int val);
}
```

- ```
JNIEXPORT void JNICALL Java_Exemple_callP
                (JNIEnv *env, jobject obj){
    jclass classe = (*env)->GetObjectClass(env,obj);
    jMethodID mid = (*env)->GetStaticMethodID(env, classe,"p", "()V");
    (*env)->CallStaticVoidMethod(env, classe,mid) ;
}
```

instruction JVM : invokestatic

cj) Types natifs

- **jni.h, interpreter.h, oobj.h, typecodes.h**
- **Types natifs / types java**

jbyte / byte, **jshort** / short, **jint** / int

....

jobject / Object, **jclass** / Class, **jstring** / String, **jarray** / array,
jthrowable / Throwable

sur la palte-forme Win32

*nous avons **typedef jint long;***

Signature des méthodes

- ***FieldType ::= BaseType | ObjectType | ArrayType***
- ***BaseType***
 - B byte
 - C char
 - D double
 - F float
 - I int
 - J long
 - S short
 - Z boolean
- ***ObjectType***
 - L<classname>;
- ***ArrayType***
 - [table
- ***MethodDescriptor ::= (FieldType *) ReturnDescriptor***
- ***ReturnDescriptor ::= FieldType | V***
 - V si le type retourné est void

Signature des méthodes en "clair"

- **>javap -s -private JavaVersC**

Compiled from JavaVersC.java

```
public synchronized class JavaVersC extends java.lang.Object
    /* ACC_SUPER bit set */
{
    public native void bonjour();
    /*    ()V    */
    public static void main(java.lang.String[]);
    /*    ([Ljava/lang/String;)V    */
    public JavaVersC();
    /*    ()V    */
    static static {};
    /*    ()V    */
}
```

Objets et classe

- **NewObject, NewObjectA, NewObjectV**

création d'une instance

obtention de la classe

obtention du constructeur

passage des paramètres et création

```
...//en Java : ClasseA newObj = new ClasseA(10, "hello");
```

```
jclass classe = (*env)->FindClass("ClasseA");  
jMethodID mid = (*env)->GetMethodID(classe,  
                                     "<init>",  
                                     "(ILjava/lang/String;)V");  
  
jint val = 10;  
jstring str = (*env)->NewStringUTF(env, "hello");  
jobject newObj = (*env)->NewObject(env,  
classe, mid, val, str) ;  
}
```

- ***instruction JVM : new et invokespecial***

instanceof

- **IsInstanceOf**

- `class A{}`

- `class B extends A{void callP(boolean b){...};}`

- *...// obj est de classe déclarée A mais constatée B*

```
jclass classeB = (*env)->FindClass("B");
if ((*env)->IsInstanceOf(obj,classeB)){
    jMethodID mid = (*env)->GetMethodID(classeB,
                                        "callP",
                                        "(Z)V");

    jbool val = JNI_TRUE;
    (*env)->CallVoidMethod(obj,mid,val) ;
}
```

instruction JVM : instanceof, (checkcast)

Tableaux et String

- **NewObjectArray,**

- ```
public class Exemple{
 public void p(){String sa = newArray(10);}
 public native String [] newArray(int taille);
}
```

- ...

```
jclass classe = (*env)->FindClass(env, "java/lang/String");
jObjectArray newArr = (*env)->NewObjectArray(env, taille,classe,NULL);
for(int i = 0; i< taille; i++){
 str = (*env)->NewStringUTF("hello");
 (*env)->SetObjectArrayElement(env, newArr,i,str);
 (*env)->DeleteLocalRef(env, str);
}
return newArr;
}
```

DeleteLocalRef -> str a 2 références, en jni et en java (gc)

***instruction JVM : newarray***

# Objets et ramasse miettes

---

- **Chaque objet crée par JNI ne peut être collecté par le ramasse miettes Java**, (l'objet str est référencé dans la machine Java)

```
DeleteLocalRef(str) // de l'exemple précédent
```

**permet de libérer cet objet** (autorise la récupération mémoire par le ramasse miettes)

```
NewGlobalRef(obj);
```

**"bloque" l'objet en mémoire**

# Levée d 'exceptions

---

- ***ThrowNew, ExceptionClear, ExceptionOccured, ExceptionDescribe***

- ...

```
jclass classeExc = (*env)->FindClass("java/lang/OutOfMemoryError");
...
if (Condition){
 (*env)->ThrowNew(classeExc, "OutOfMemoryError");

 printf("apres le traitement de l'exception en Java ...");
 ...
}
```

- ***instruction JVM : athrow***

# Monitor

---

- **MonitorEnter, MonitorExit**

```
// l'équivalent de l'instruction synchronized
(*env)->MonitorEnter(env,obj);

//du code C/C++

(*env)->MonitorExit(env,obj);
}
```

## ***instructions JVM : monitorenter, monitorexit***

Appels de **wait** et **notify** par les primitives GetMethodID et CallVoidMethod

```
jclass classe = (*env)->GetObjectClass(env,obj);
jMethodID waitMid = (*env)->GetMethodID(env,classe,"wait","()V");
(*env)->CallVoidMethod(env,obj,waitMid);
if((*env)->ExceptionOccured()!=NULL)
 // une exception est générée en Java, mauvais usage de Wait ...
}
```

# Deux exemples

---

- **Entrées/Sorties Voie Série**
- **Accès au « Windows Registry »**
- ...



# Entrées/Sorties Voie Série

---

- **Win32Port Markus Bauer, (disponible sur <http://java.cnam.fr>)**

- `public class Win32Port{`

- `public native boolean open(int pnr, String par, boolean handshake);`
- `public native void clear(int pnr);`
- `public native void setTimeout(int pnr, int tout);`
- `public native int getByte(int pnr);`
- `public native int getBytes(int pnr, byte[] bytes);`
- `public native boolean putByte(int pnr, int b);`
- `public native int putBytes(int pnr, byte[] bytes);`
- `public native boolean close(int pnr);`

- `static {`
- `System.loadLibrary("portcom");`
- `}}`

# Accès à la base de registres sous Windows

---

- <http://www.emunix.emich.edu/~evett/ProgrammingLanguages/Core/v2ch11/Win32RegistryTest/>
- **Un lien à suivre ... et à mettre en Oeuvre**

# Appel de la machine Java

---

- **le source C *ordinaire***

  - utilisation de `javai.lib`

  - chargement et appel de la machine Java depuis le point d'entrée `main`

- **le source Java *ordinaire***

  - n'importe quelle application

# Le source C

---

```
#include <jni.h>

int main(int argc, char *argv[]){
 // declarations ici
 options[0].optionString = "-Djava.class.path=";
 memset(&vm_args, 0, sizeof(vm_args));
 vm_args.version = JNI_VERSION_1_2;
 vm_args.nOptions = 1;
 vm_args.options = options;

 res = JNI_CreateJavaVM(&jvm, (void**)&env, &vm_args);
 res = JNI_CreateJavaVM(&jvm, &env, &vm_args);
 classe = (*env)->FindClass(env, "CVersJava");
 methodeID = (*env)->GetStaticMethodID(
 env, classe, "main", "([Ljava/lang/String;)V");

 jstr = (*env)->NewStringUTF(env, " depuis du c !!!");
 args = (*env)->NewObjectArray(
 env, 1, (*env)->FindClass(env, "java/lang/String"), jstr);
 (*env)->CallStaticVoidMethod(env, classe, methodeID, args);
 (*jvm)->DestroyJavaVM(jvm);
} return (0);}
```

# Le source Java

---

- `public class CVersJava{`
- 
- `public static void main(String [] args){`
- `System.out.println("en java dans main " + args[0]);`
- `}`
- `}`

# Le source Complet (1)

---

```
#include <jni.h>
#include <stdlib.h>

int main(int argc, char *argv[]){
 JNIEnv *env;
 JavaVM *jvm;
 JavaVMInitArgs vm_args;
 JavaVMOption options[1];

 jint res;
 jclass classe;
 jmethodID methodID;
 jstring jstr;
 jobjectArray args;
 char classpath[1024];

 options[0].optionString = "-Djava.class.path=";
 options[1].optionString = "-verbose:jni";
 memset(&vm_args, 0, sizeof(vm_args));
 vm_args.version = JNI_VERSION_1_2;
 vm_args.nOptions = 1;
 vm_args.options = options;
```

# Le source Complet (2)

---

```
res = JNI_CreateJavaVM(&jvm, (void**)&env, &vm_args);
if (res==JNI_ERR){
 printf(" erreur lors de la création de la JVM\n");
 return 1;
}
classe = (*env)->FindClass(env, "CVersJava");
methodeID = (*env)->GetStaticMethodID(env,classe,"main","([Ljava/lang/String;)V");

jstr = (*env)->NewStringUTF(env," depuis du c !!!");
args = (*env)->NewObjectArray(env,1,(*env)->FindClass(env,"java/lang/String"),jstr);
(*env)->CallStaticVoidMethod(env,classe,methodeID,args);
(*jvm)->DestroyJavaVM(jvm);

return (0);
}

cl -lc:\jdk\include -lc:\jdk\include\win32 SimpleHttpd.c c:\jdk\lib\jvm.lib
set path = c:\jdk\jre\bin\classic\;%path% accès à jvm.dll
```

# Un serveur Web depuis C ...

---

....

```
res = JNI_CreateJavaVM(&jvm, (void**)&env, &vm_args);
if (res==JNI_ERR){
 printf(" erreur lors de la création de la JVM\n");
 return 1;
}
classe = (*env)->FindClass(env, "SimpleHttpd2");
methodeID = (*env)->GetStaticMethodID(env,classe,"main","([Ljava/lang/String;)V");
args = (*env)->NewObjectArray(env,0,(*env)->FindClass(env,"java/lang/String"),NULL);

(*env)->CallStaticVoidMethod(env,classe,methodeID,args);

return (0);
}
```



# Conclusion

---