

# TP 10 : JNI et langage C/Java

JNI (Java Native Interface) est une bibliothèque logicielle d'interfaçage qui permet au code Java s'exécutant à l'intérieur de la JVM (Java Virtual Machine) d'appeler et d'être appelé par des applications natives (c'est-à-dire des programmes spécifiquement liés au matériel et au système d'exploitation de la plate-forme concernée), ou avec des bibliothèques logicielles basées sur d'autres langages (C, C++, assembleur, etc.).

Il faut installer tout d'abord le JDK Java et instancier la variable **JAVA\_HOME** :

```
sudo apt install openjdk-14-jdk-headless
export JAVA_HOME=/usr/lib/jvm/java-14-openjdk-amd64
```

## 1. Un premier exemple

### 1.1 Etape 1 : Créer le fichier java

Dans un premier temps, créons le fichier `HelloJNI.java` (cf. ci-dessous).

```
package fr.ut3;
class HelloJNI {
    static {System.loadLibrary("Greetings");} // librairie native

    public static native String getGreetings(String who);

    public static void main(String[] args) {
        HelloJNI h = new HelloJNI();
        System.out.println(getGreetings(args[0]));
    }
}
```

### 1.2 Etape 2 : compiler le fichier java

Dans un 2<sup>ème</sup> temps, compilons le fichier (attention à bien utiliser l'option -d)

```
javac -d . HelloJNI.java
```

Normalement, le fichier `.class` est créé dans les sous-répertoires : `fr/ut3/HelloJNI.class`

### 1.3 Etape 3 : générer le fichier .h

L'usage de la commande suivante va permettre de générer les headers `.h` (à ne pas modifier) qui vont permettre de coder les fonctions en C et/ou C++

```
javac -h . HelloJNI.java (pour Java 8 et supérieur) qui va générer le code fr_ut3_HelloJNI.h
```

**Code généré :**

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class fr_ut3_HelloJNI */

#ifdef _Included_fr_ut3_HelloJNI
#define _Included_fr_ut3_HelloJNI
#ifdef __cplusplus
extern "C" {
#endif

/*
 * Class:      fr_ut3_HelloJNI
 * Method:     getGreetings
 * Signature:  (Ljava/lang/String;)Ljava/lang/String;
 */
JNIEXPORT jstring JNICALL Java_fr_ut3_HelloJNI_getGreetings
    (JNIEnv *, jclass, jstring);

#ifdef __cplusplus
}
#endif
#endif
```

**1.4 Etape 4 : créer le fichier .c**

Enfin, il faut coder le programme .c qui va implémenter les fonctions natives (par exemple dans le fichier Greetings.c

```
// Greetings.c
#include <jni.h>
#include "fr_ut3_HelloJNI.h"

JNIEXPORT jstring JNICALL Java_fr_ut3_HelloJNI_getGreetings
    (JNIEnv* env, jclass obj, jstring string) {
    const char* str = (*env)->GetStringUTFChars(env, string, 0);
    char cap[128];
    // strcpy(cap, str);
    sprintf(cap, "You welcome, %s\n", str);
    (*env)->ReleaseStringUTFChars(env, string, str);
    return ((*env)->NewStringUTF(env, cap));
}
```

**1.5 Générer la librairie dynamique**

Dernière étape, générons la librairie dynamique (.so sous Unix, .dll sous windows) grâce aux deux commandes suivantes :

```
gcc -c -I"$JAVA_HOME/include" -I"$JAVA_HOME/include/linux" -o Greetings.o Greetings.c
gcc -shared -o libGreetings.so Greetings.o
```

**1.6 Exécuter le programme**

Enfin il suffit d'exécuter le programme en donnant en argument où trouver les librairies natives. Normalement, un message s'affiche 😊

```
java -Djava.library.path=. fr.ut3.HelloJNI Guy
```

## 2. Les transtypages

Le mapping entre les types java et ceux des méthodes natives est assez simple. Le schéma de nommage reste à peu les mêmes : les types natifs sont précédés du caractère « j » suivi par le nom en minuscules équivalent en java. JNI inclus un autre type nommé « *jsize* » qui stocke la longueur d'un tableau ou d'une chaîne de caractères.

Type de données Java	Type de données natif	Description
Void	void	Type vide
Byte	jbyte	8 bits signés. Valeurs entre $-2^7$ à $2^7 - 1$
Int	jint	32 bits signés. Valeurs entre $-2^{31}$ et $2^{31} - 1$
float	jfloat	32 bits. Représente une valeur réelle entre $1,4 \cdot 10^{45}$ et $3,4 \cdot 10^{38}$ (approx.), positive ou négative
double	jdouble	64 bits. Représente une valeur réelle entre $4,9 \cdot 10^{324}$ et $1,7 \cdot 10^{308}$ (approx.), positive ou négative
char	jchar	16 bits non signés. Valeur entre 0 et 65535
long	jlong	64 bits signés. Valeur entre $-2^{63}$ et $2^{63} - 1$
short	jshort	16-bit signés. Valeur entre $-2^{15}$ et $2^{15} - 1$
boolean	jboolean	8 bits non signés. <i>true</i> et <i>false</i>

En outre, JNI définit d'autres références classiques comme les chaînes de caractères, classes, ... D'autres références peuvent aussi être mappées et manipulables avec *jobject*.

Type de Référence Java	Type JNI	Description
java.lang.Object	jobject	N'importe quel objet Java
java.lang.String	jstring	Représentation des chaînes de caractères
java.lang.Class	jclass	classe objet Java
java.lang.Throwable	jthrowable	Objet Java levant une exception

### 3. A vous de jouer !!!

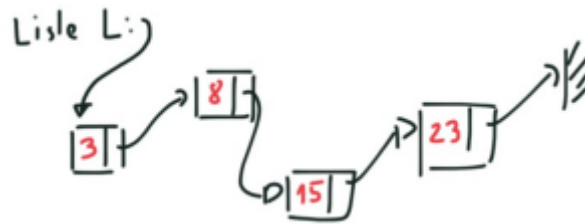
A vous maintenant de créer votre interface JNI.

Vous devrez gérer une liste de valeurs entières triées dans une **liste chaînée en C**. Chaque cellule de la liste possède deux champs :

1. un champ *valeur* pour stocker une valeur entière,
2. et un champ *suivant* pour stocker l'adresse de la cellule suivante.

On s'intéresse ici aux listes triées dans l'ordre croissant : on sait donc que le premier élément de la liste est le plus petit, le second est le deuxième plus petit, etc.

Par exemple, la liste suivante est correcte :



Votre programme Java devra implémenter les fonctions natives suivantes :

- une fonction **creerListe()** qui crée une liste
- la fonction **ajouter(entier x)** qui ajoute une cellule (au bon endroit pour que la liste reste triée) dans la liste
- une fonction **supprimer(entier x)** qui supprime une valeur de la liste
- une fonction **tester(entier x)** qui renvoie vrai si il existe une cellule dans L contenant la valeur x, et faux sinon.
- Et une fonction **compter()** qui compte le nombre d'éléments de la liste

Vous écrierez enfin un programme java appelant ces fonctions natives (en utilisant ou non une interface graphique)