Programming Technologies: Assignment
The Java Native Interface Programming

Due on Noviembre 2016

José A. Aviña - JNI Programming

Índice

Assignment 1

Compilar y ejecutar los programas en listados 1 y 2 para embeber la Máquina Virtual de Java en Ansi C. Los códigos fuentes de las Clases de Java para miprog-c.c y jni-test.c, los encontrarán anexos en el EDMODO.

Listing 1: miprog-c.c Ansi C source code

```
#include <jni.h>

#define PATH_SEPARATOR ':'

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

   long status;
   jclass cls;
   jmethodID mid;
   jint square;
   jboolean not;

   options[0].optionString = "-Djava.class.path=.";

   //(memset(&vm_args, 'c', sizeof(vm_args));


   vm_args.version = JNI_VERSION_1_2;
   vm_args.nOptions = 1;
   vm_args.options = options;

   status = JNI_CreateJavaVM(&jvm, (void**)&env, &vm_args);

   if (status != JNI_ERR)
   {
     cls = (*env)->FindClass(env, "MiClase");
     if(cls !=0)
     { mid = (*env)->GetStaticMethodID(env, cls, "intMethod", "(I)I");
       if(mid !=0)
       { square = (*env)->CallStaticIntMethod(env, cls, mid, 2);
         printf("Result of intMethod: %d\n", square);
       }

       mid = (*env)->GetStaticMethodID(env, cls, "booleanMethod", "(Z)Z");
       if(mid !=0)
       { not = (*env)->CallStaticBooleanMethod(env, cls, mid, 1);
         printf("Result of booleanMethod: %d\n", not);
       }
     }

     (*jvm)->DestroyJavaVM(jvm);
     return 0;
```

Assignment 1 continued on next page. . .

```
  }
  else
  return -1;
}
```

Listing 2: jni-test.c Ansi C source code

```
/* Remember to include this */
#include <jni.h>

/* This is a simple main file to show how to call Java from C++ */

int main()
{
  /* The following is a list of objects provided by JNI.
     The names should be self-explanatory...
   */
  JavaVMOption options[1]; // A list of options to build a JVM from C++
  JNIEnv *env;
  JavaVM *jvm;
  JavaVMInitArgs vm_args; // Arguments for the JVM (see below)

  // This will be used to reference the Java class SimpleJNITest
  jclass cls;

  // This will be used to reference the constructor of the class
  jmethodID constructor;

  // This will be used to reference the object created by calling
  // the constructor of the class above:
  jobject simpleJNITestInstance;

  // You may need to change this. This is relative to the location
  // of the C++ executable
  options[0].optionString = "-Djava.class.path=.";

  // Setting the arguments to create a JVM...
  //memset(&vm_args, 0, sizeof(vm_args));
  vm_args.version = JNI_VERSION_1_2;
  vm_args.nOptions = 1;
  vm_args.options = options;


  long status = JNI_CreateJavaVM(&jvm, (void**)&env, &vm_args);

  if (status != JNI_ERR) {
    // If there was no error, let's create a reference to the class.
    // Make sure that this is in the class path specified above in the
    // options array
    cls = (*env)->FindClass(env, "SimpleJNITest");

    if(cls !=0) {
      printf("Class found \n");
      // As above, if there was no error...
```

```
        /* Let's build a reference to the constructor.
        This is done using the GetMethodID of env.
        This method takes
        (1) a reference to the class (cls)
        (2) the name of the method. For a constructor, this is
               <init>; for standard methods this would be the actual
               name of the method (see below).
        (3) the signature of the method (its internal representation, to be precise).
            To get the signature the quickest way is to use javap. Just type:
               javap -s -p com/example/simple/SimpleJNITest.class
               And you will get the signatures of all the methods,
               including the constructor (check the "descriptor" field).
               In the case of the constructor, the signature is "()V", which
               means that the constructor does not take arguments and has no
               return value. See below for other examples.
         */
        constructor = (*env)->GetMethodID(env, cls, "<init>", "()V");

        if(constructor !=0 ) {

       printf("Constructor found \n");
       /* If there was no error, let's create an instance of the SimpleTestJNI by
          calling its constructor
       */
       jobject simpleJNITestInstance = (*env)->NewObject(env, cls, constructor);

       if ( simpleJNITestInstance != 0 ) {
         /* If there was no error, let's call some methods of the
            newly created instance
         */

         printf("Instance created \n");

         // First of all, we create two Strings to be passed
         // as argument to the addValue method.
         jstring jstr1 = (*env)->NewStringUTF(env, "Hola Mundo");
         jstring jstr2 = (*env)->NewStringUTF(env, "JNI");

         /* Then, we create a reference to the addValue method.
            This is very similar to the reference to the constructor above.
            As above, you can get the signature of the addValue method with javap.
            In this case, the method takes a String as input and does not return
            a value (V is for void)
          */

         jmethodID addValue = (*env)->GetMethodID(env, cls, "addValue", "(Ljava/lang/String;)V");

         /* Finally, let's call the method twice, with two different arguments.
            (it would probably be a good idea to check for errors here... This is
             left as a simple exercise to the student ;-).
          */
         (*env)->CallObjectMethod(env, simpleJNITestInstance , addValue, jstr1);
         (*env)->CallObjectMethod(env, simpleJNITestInstance , addValue, jstr2);

         /* Let's now call another Java method: printValues should print on screen
            the content of the List of values in the object. The pattern is identical
            to the one above, but no arguments are passed.
         */
         jmethodID printValues = (*env)->GetMethodID(env, cls, "printValues", "()V");
         (*env)->CallObjectMethod(env, simpleJNITestInstance , printValues);

         /* Finally, let's extract an int value from a method call. We need to
            invoke CallIntMethod and we are going to use getSize of
            simpleJNITestInstance. Notice the signature: the
            method returns an int and it does not take arguments
         */
         jint listSize;
         jmethodID getSize = (*env)->GetMethodID(env, cls, "getSize", "()I");
         listSize = (*env)->CallIntMethod(env, simpleJNITestInstance , getSize);

         printf("The size of the Java list is: %d\n", listSize);


          }
        }
        else {
       printf("I could not create constructor\n");
        }
      }
      (*jvm)->DestroyJavaVM(jvm);
      printf("All done, bye bye!\n");
      return 0;
    }
    else
      return -1;
  }
```

Assignment 2

Implementar una aplicación en Ansi C/JESS para diagnosticar dos cuadros patológicos a partir de sus respectivos síntimas. La selección de los cuadros patalógicos por diagnóticar es libre y/o abierta. La aplicación debe:

- Inicializar la Memoria de Trabajo (The Working Memory).
- Registrar reglas de producción (defrule r1 . . .).
- Aceverar síntomas conocidos (assert . . .).
- Invocar la máquina RETE de razonamiento (run)