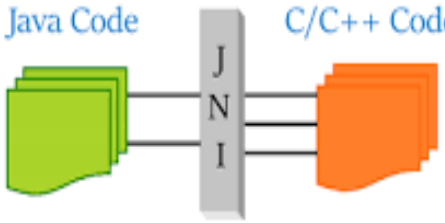
# Sample Dll program as follow

- we declare the method `sayHello()` as a native instance method, via keyword `native` which denotes that this method is implemented in another language. A native method does not contain a body. The `sayHello()` shall be found in the native library loaded.
- The `main()` method allocates an instance of `HelloJNI` and invoke the native method `sayHello()`.




---

## Step 1: Write a Java Class HelloJNI.java that uses C Codes

```
    public class HelloJNI {  // Save as HelloJNI.java
1      static {
2         System.loadLibrary("hello");
3   // Load native library hello.dll (Windows) or libhello.so (Unixes)
4                                      //  at runtime
5                           // This library contains a native method called sayHello()
6      }
7
8      // Declare an instance native method sayHello() which receives no parameter
9   //and returns void
10     private native void sayHello();
11
12     // Test Driver
13     public static void main(String[] args) {
14        new HelloJNI().sayHello();  // Create an instance and invoke the native method
15     }
    }
```

## Step 2: Implementing the C Program HelloJNI.c

```
1                           // Save as "HelloJNI.c"
2   #include <jni.h>        // JNI header provided by JDK
3   #include <stdio.h>      // C Standard IO Header
4   #include "HelloJNI.h"   // Generated
5
6   // Implementation of the native method sayHello()
7   JNIEXPORT void JNICALL Java_HelloJNI_sayHello(JNIEnv *env, jobject thisObj) {
8      printf("Hello World!\n");
9      return;
10  }
```

# Steps to run -

**> javac HelloJNI.java**

**>javah -jni HelloJNI**

**> gcc -I/usr/lib/jvm/java-7-openjdk-amd64/include -I/usr/lib/jvm/java-7-openjdk-amd64/include/linux -o libhello.so -shared -fPIC HelloJNI.c**

**> java -Djava.library.path=.  HelloJNI**

**Note-** to find path of include folder for ur machine use  following command
> locate jni.h

---

# Assignment : Write a program to create Dynamic Link Library for any mathematical operation and write  an application program to test it. (Java Native Interface / Use VB or VC++).

## Steps 1) – save prog as B1.c

```
#include <jni.h>
 #include <stdio.h>
 #include "B1.h"
 JNIEXPORT int JNICALL Java_B1_add(JNIEnv *env, jobject obj, jint a, jint b)
 {
         printf("\n%d + %d = %d\n",a,b,(a+b));
         return;
 }

 JNIEXPORT int JNICALL Java_B1_sub(JNIEnv *env, jobject obj, jint a, jint b)
 {
    printf("\n%d - %d = %d\n",a,b,(a-b));
    return;
 }
```

```
JNIEXPORT int JNICALL Java_B1_mult(JNIEnv *env, jobject obj, jint a, jint b)
{
   printf("\n%d * %d = %d\n",a,b,(a*b));
   return;
}

JNIEXPORT int JNICALL Java_B1_div(JNIEnv *env, jobject obj, jint a, jint b)
{
   printf("\n%d / %d = %d\n",a,b,(a/b));
   return;
}
```

99999999999999999999999999999999999999999999999

# steps 2- save program as b1.java

```
import java.io.*;
import java.util.*;
class B1 {
     static {
        System.loadLibrary("B1");
    }
    private native int add(int a, int b);
    private native int sub(int a, int b);
    private native int mult(int a, int b);
    private native int div(int a, int b);
     public static void main(String[] args) {
          Scanner sc=new Scanner(System.in);
          int a, b,ch;
          System.out.println("\nEnter value of a : ");
          a = sc.nextInt();
          System.out.println("\nEnter value of b : ");
          b = sc.nextInt();
          do
          {
                    System.out.println("\nENTER YOUR CHOICE : ");
                    ch = sc.nextInt();
          switch(ch)
                    {
                              case 1 : new B1().add(a,b);
                                        break;
                              case 2 : new B1().sub(a,b);
                                        break;
                              case 3 : new B1().mult(a,b);
                                        break;
                              case 4 : new B1().div(a,b);
                                        break;
                              default : System.out.println("Your choice is wrong.");
          }
          }while(ch<5);
    }
}
```

## steps to run-

**> javac B1.java**
**>javah -jni B1**

**>gcc -I/usr/lib/jvm/java-7-openjdk-amd64/include -I/usr/lib/jvm/java-7-openjdk-amd64/include/linux -o libB1.so -shared -fPIC B1.c**
**> java -Djava.library.path=. B1**

---

# points to notes:

“native” keyword – as we've already covered, any method marked as native must be implemented in a native, shared lib.

- System.loadLibrary(String libname) – a static method that loads a shared library from the file system into memory and makes its exported functions available for our Java code.

C/C++ elements (many of them defined within jni.h)

- JNIEXPORT- marks the function into the shared lib as exportable so it will be included in the function table, and thus JNI can find it
- JNICALL – combined with JNIEXPORT, it ensures that our methods are available for the JNI framework
- JNIEnv – a structure containing methods that we can use our native code to access Java elements
- JavaVM – a structure that lets us manipulate a running JVM (or even start a new one) adding threads to it, destroying it, etc…

## for more info refer

## https://www.baeldung.com/jni

-