Announcements

- **Program #5**
  - is due today
- **Program #6**
  - Is on the web
- **Reading**
  - Today – Notes
  - Tuesday - Notes

Java Native Interface

- **Goals**
  - Sometimes need to access things in lower level language
  - Provide full access to Java Objects in other languages
- **Keys Ideas**
  - Need to identify signatures of native methods
  - Need to incorporate native code into program
JNI definition

- The JNI defines a standard naming and calling convention so the Java virtual machine can locate and invoke native methods.
- JNI is built into the Java virtual machine so the Java virtual machine can invoke local system calls to perform input and output, graphics, networking, and threading operations on the host operating system.
- Call code written in any programming language from a program written in the Java language
  - by declaring a native Java method
  - loading the library that contains the native code
  - and then calling the native method

Process of Compiling Using JNI

1. **Helloworld.java** → **Java Compiler (javac)** → **Helloworld.class** → **Javah -jni** → **Helloworld.h** → **gcc** → **Hello.so**
2. **HelloworldC.c** → **jni.h**
3. **stdio.h**
Using JNI Functions

- **Declaring Functions**
  - Declare like a normal method, but add “native” keyword
  - public native void displayHelloWorld();

- **Loading the Code**
  - Use loadLibrary call
  - Making it static forces it to happened automatically
  - System.loadLibrary("hello")

- **Invoke Native Method**
  - new HelloWorld().displayHelloWorld();

Full Example of JNI

- **Java Code**
  ```java
class HelloWorld {
    public native void displayHelloWorld();
    static { System.loadLibrary("hello"); }
    public static void main(String[] args) {
      new HelloWorld().displayHelloWorld();
    }
  }
  ```

- **C Function Signature**
  - Required Keywords JNIEXPORT JNICALL
  - Constructing the function name
  - Java_ + Full Classname + _ + Method Name

  ```c
  JNIEXPORT void JNICALL Java_HelloWorld_displayHelloWorld(JNIEnv *, jobject);
  ```
C Code for JNI Call

```c
#include <jni.h>
#include "HelloWorld.h"
#include <stdio.h>

JNIEXPORT void JNICALL Java_HelloWorld_displayHelloWorld(JNIEnv *env, jobject obj)
{
    printf("Hello world!\n");
    return;
}
```

Accessing Java Data from C

- Mapping Types

<table>
<thead>
<tr>
<th>Java Type</th>
<th>Native Type</th>
<th>Size (bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>Jboolean</td>
<td>8, unsigned</td>
</tr>
<tr>
<td>Byte</td>
<td>Jbyte</td>
<td>8</td>
</tr>
<tr>
<td>Char</td>
<td>Jchar</td>
<td>16</td>
</tr>
<tr>
<td>Short</td>
<td>Jshort</td>
<td>16</td>
</tr>
<tr>
<td>Int</td>
<td>Jint</td>
<td>32</td>
</tr>
<tr>
<td>Long</td>
<td>Jlong</td>
<td>64</td>
</tr>
</tbody>
</table>
Java Strings

- The String object in the Java language, which is represented as jstring in Java Native Interface (JNI), is a 16 bit unicode string.
- In C a string is by default constructed from 8 bit characters.

- To access a Java language String object passed to a C or C++ function or return a C or C++ string to a Java language method, you need to use JNI conversion functions in your native method implementation.

- The following converts the java string to an array of C characters:
  - (*env)->GetStringUTFChars(env, name, iscopy)

- The following C JNI function converts an array of C characters to a jstring:
  - (*env)->NewStringUTF(env, lastfile)

Access Java Arrays

- Need to call special functions to get data and size
- Example:

```c
JNIEXPORT jfloat JNICALL Java_FloatArray_sumArray(JNIEnv *env, jobject obj, jfloatArray arr)  {
    jfloat *body, sum = 0;
    jsize I, len;

    len = (*env)->GetArrayLength(env, arr);
    body = (*env)->GetFloatArrayElements(env, arr, 0);
    for (i=0; i<len; i++) {
        sum += body[i];
    }
    (*env)->ReleaseFloatArrayElements(env, arr, body, 0);
    return sum;
}
```
Calling Java Methods from C

- **Steps**
  - Get Object handle
  - Get MethodID – need to know signature
    - (argument-types)return-type
  - Call method
- **Example**
  ```c
  JNIEXPORT void JNICALL Java_Callbacks_nativeMethod(
    JNIEnv *env, jobject obj, jint depth) {
    jclass cls = (*env)->GetObjectClass(env, obj);
    jmethodID mid = (*env)->GetMethodID(env, cls, "callback", "(I)V");
    if (mid == 0) { return; }
    printf("In C, depth = %d, about to enter Java\n", depth);
    (*env)->CallVoidMethod(env, obj, mid, depth);
    printf("In C, depth = %d, back from Java\n", depth);
  }
  ```

Accessing Object Fields

- **Getting FieldID**
  - Need to get a unique id for the field
  - Must specify type information
    - fid = (*env)->GetFieldID(env, cls, "s", "Ljava/lang/String;");
      - cls is of type jclass
      - S is field name
  - Can make one call to GetFieldID
    - Really computes offset for item
- **Getting the data**
  - Pass the fid, env, and object
    - str = (*env)->GetObjectField(env, obj, fid);