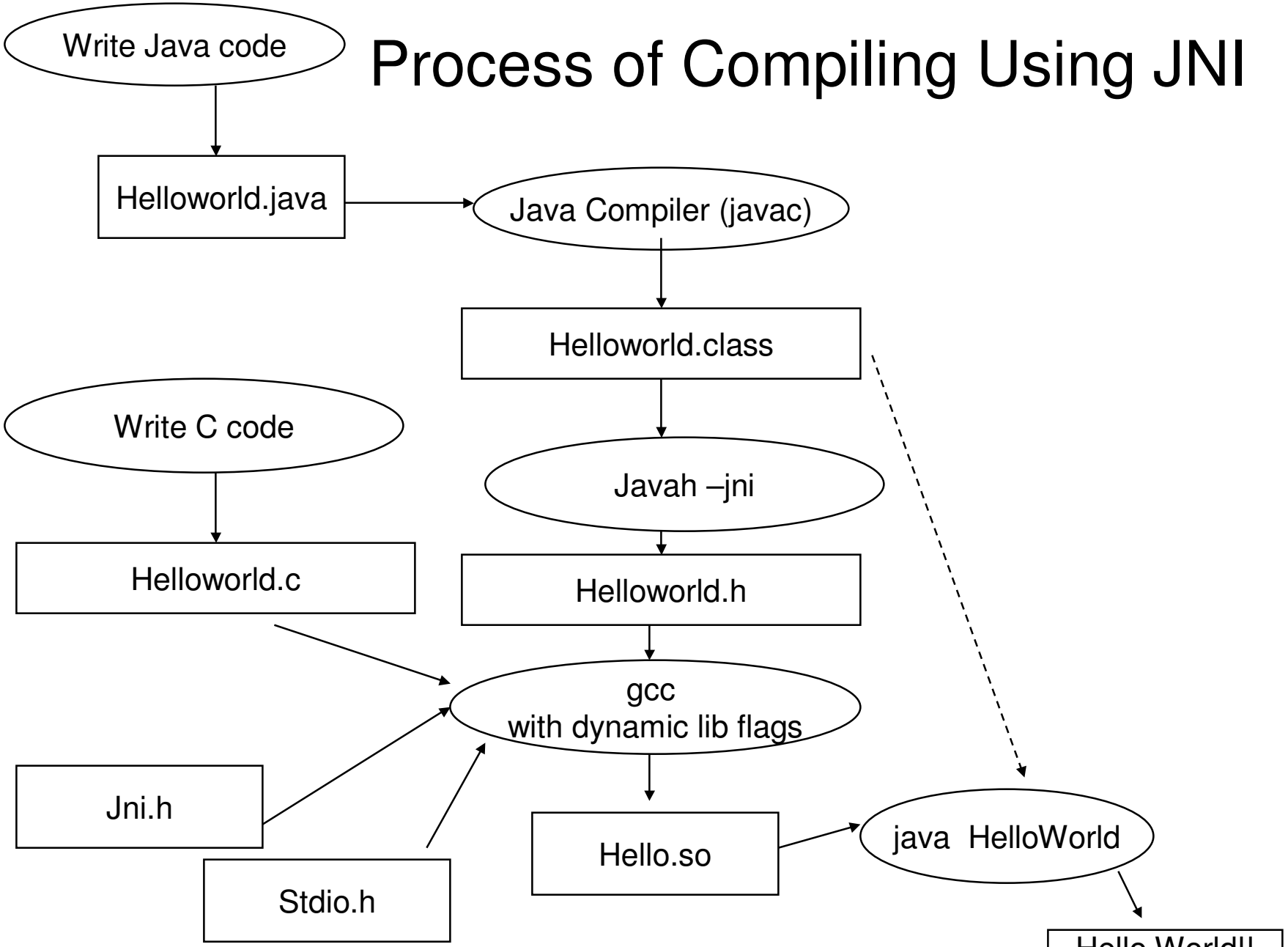
Announcements

- Program #5
  - is due Thursday
- Reading
  - http://java.sun.com/docs/books/tutorial/native1.1/
- Pickup old assignments with project #4
  - Midterms (including re-grades)
  - Programming assignments

Java Native Interface

- Goals (both directions)
  - 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

Process of Compiling Using JNI



Using JNI Functions

- Declaring Functions in Java
  - (where the function to be used is written in C)
  - 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")
  - Java’s equivalent of dlopen
  - adds the appropriate name completion for this system
- Invoke Native Method
  - new HelloWorld().displayHelloWorld();
  - this invokes the C code

```
class HelloWorld {
        public native
                void displayHelloWorld();

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

        public static void main(String[] args)
        {
          new
            HelloWorld().displayHelloWorld();
         }
}
```

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

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

C Function Signature-Constructing the function name

- Java_ + Full Classname + _ + Method Name

```
JNIEXPORT void JNICALL Java_HelloWorld_displayHelloWorld (JNIEnv *, jobject);
```

Accessing Java Data from C

- Mapping Types

Java Type	Native Type	Size (bits)
Boolean	Jboolean	8, unsigned
Byte	Jbyte	8
Char	Jchar	16
Short	Jshort	16
Int	Jint	32
Long	Jlong	64

Access Java Arrays

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

```
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

```
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);