Using the Java Native Interface to Introduce Device Driver Basics

James K. Brown, Indiana University-Purdue University at Indianapolis

ABSTRACT

Teaching students to merge the real-world necessity of a graphical user interface (GUI) with the old-world techniques of manipulating the computer hardware can be easily accomplished using the Java Native Interface. The GUI capabilities can be acquired using the strongly typed and completely object oriented programming language of Java. Java allows users to quickly build user interfaces in a graphical form and has a very powerful tool in the Java Native Interface (JNI) to allow access to lower-level (native) programs. This interface is needed because Java runs on top of a virtual machine (VM). This allows the language to be platform independent but then inhibits a user's ability to access various hardware inputs and outputs (I/O). To use a simple I/O port such as the serial port on a PC, a Java program must leave the VM and execute corresponding "native" code to manipulate the USART. The students can then write the native code in C and they gain multiple skills that can be taken to the workplace.

This paper presents some of the methodology and ideology behind using Java, the JNI and the C programming language to expose students to the rigors of basic device driver development. This will be done with reference to the Linux operating system as it allows students to easily grasp at the system level what is taking place in the hardware.

INTRODUCTION

Allowing a "real world" experience into the classroom is sometimes challenging – especially for computer programming classes that usually focus more on style and syntax rather than addressing needs that might be seen in industry.

Today's technology expectations and cost drivers force the programmer/engineer to be able to provide more capability in a graphical format and prototype models and interfaces in a shorter development cycle.

This paper will focus on classroom instruction that exposes the student to both object oriented programming (OOP) skills, low level device driver techniques and the interface between the two. The OOP language used in this course is Java, the device driver development language is C, and the interface between the two is the Java Native Interface. These three capabilities are combined in one course as an attempt to teach introductory techniques in device driver development.

The Java discussions will center upon the need to graphically display laboratory or field data on a desktop PC. While not comprehensive, the language topics used in coursework to provide useful capability to the student will be examined and their focus will remain on designing a basic

"Proceedings of the Spring 2007 American Society for Engineering Education Illinois-Indiana Section Conference.
Copyright © 2007, American Society for Engineering Education"

graphical user interface that can be used for the rest of the class by making only minor alterations of the look and feel.

The C programming portion of the class is designed to quickly take the student from being able to print out "Hello World" to using pointers to parse strings and also to perform file operations. This quick up-to-speed approach is used to portray the C programming as only a necessary capability in the quest to understand and fully utilize the Java Native Interface, or JNI.

The JNI, by necessity, demands significant attention by the student as it is the "tool" that allows the students to access the hardware ports, address and devices of a desktop computer from Java.

One lab project in which the students are asked to develop a serial port driver using Java as the graphical user interface, C programming for the "driver" and the JNI as the "go-between" in the system is described. This lab is an attempt to not only bolster their programming confidence but to also bring the "real-world" into their programming skills.

COURSE CONTENT

Java Programming

The semester course consists of 15 weeks of instruction and a final exam week. The course is roughly divided into two halves; the first half consisting of instruction and labs that bring object oriented programming techniques to the grasp of the student and the second half introducing the C programming language in conjunction with labs of increasing complexity.

The Java programming section is designed to not only teach the fundamentals of object oriented programming and the Java programming language but to also bring a corresponding programming confidence. This is accomplished by beginning the course with fundamental OOP concepts, asking the students to apply these concepts in a team based experience and then introducing the Java language as a simple extension of the OOP. Topics covered in the Java section include: classes, objects, methods, constructors (all normal OOP topics) and then the course advances into the Swing components (Frames, Panes, dialog boxes, etc.), layout managers and exception handling. Finally, the Java section concludes with input/output (I/O), networking and multi-threading which somewhat conclude the fundamental "toolbox" for the student and allow him/her to provide a basic graphical user interface that can be adapted for most situations.

The labs during this time period also increase in difficulty from the basic "Hello World" to designing and implementing a "Messenger" type of client-server chat program using TCP/IP that must be demonstrated over the intranet. The final result can be seen in Figure 1.

Copyright © 2007, American Society for Engineering Education"

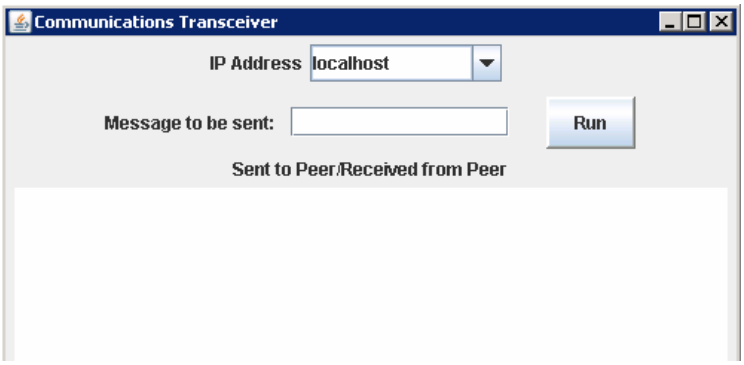

Figure 1 – Client-Server "Chat" Interface

C Programming

The second half of the semester covers both C programming and the Java Native Interface (JNI). The C programming portion takes the student through the basics of file I/O (and console I/O), pointers, arrays and structures. These are all necessary topics as the students must (for the lab assignments) search arrays and set up structures for timing delays as well as assess the needs that correspond to USART setup when implementing operations with the serial port. Other labs in which the student must perform C programming include finding and passing the MAC address to the GUI from native code, writing to a 1X8 character LCD using the parallel port and reading a Data Acquisition board over the USB bus.

Java Native Interface

The very useful and non-traditional part of this programming course is the Java Native Interface (JNI). The name Java "Native" Interface implies to the reader but not necessarily the student that this interface allows the use of "native" code (code that is not running on the virtual machine) by interfacing it to the higher level graphical application. Students, especially those who have not worked outside of the educational system, usually have not experienced "legacy" code and so do not easily discern the power they have in updating industrial plants or even data acquisition functions. Existing legacy code written in C and potentially other lower level languages can be accessed by Java and exploited in a way that is transparent to the user and thus extend the versatility and value of Java to companies that desire to update their software products.

For this course, however, the student is required to write their own native code. These pieces of native code, which are the results of their C programming labs, are then interfaced to their previously designed graphical user interfaces and the resulting integration efforts provide the valuable experiences that students sometimes miss out on in traditional coursework.

The JNI, which at first glance seems complex, is a somewhat straightforward interface that is used in a methodical 5-step manner. Assuming the steps are implemented sequentially, the JNI is easily mastered and provides the student with a powerful and useful tool that they may take out of the university system with them and allow them to contribute in and employment environment that may not always be encountered by employers hiring new graduates.

Copyright © 2007, American Society for Engineering Education"

The five steps to implement the JNI (with sample code) are as follows:

1. Declare the native method in the Java program using the keyword native (no implementation is necessary at this point).

```
public class RS232
{
native String Receiver();
native void Transmitter(String sendtxt);
```

2. Use the static directive of System.loadlibrary is included in the Java program to ensure that Java loads the needed library.

```
static {System.loadLibrary("libRS232.so");}
```

3. Run the javah utility to generate the method (function) names that must be used in for the corresponding native C program.

```
/root>> javac SerialPortLab.java
/root >>javah SerialPortLab
```

4. Implement the low-level application method(s) in C.

```
JNIEXPORT void JNICALL Java_RS232_Transmitter(JNIEnv *env, jobject, jstring text)
{
static int serialport_fd;
static FILE* serialPort;

// Initialize the serial port
// Write to the serial port then flush & close it
}// end transmitter
```

5. Compile the corresponding native code as a shared library (it must be dynamically linked).

```
g++ -I/opt/jdk1.5.0_01/include -I/opt/jdk1.5.0_01/include/linux RS232.cpp -fPIC -shared -Wl,-soname,lib
RS232.so -o libRS232.so
```

Once the five steps have been successfully completed, the student is able, in the serial port lab, to write to and read from the serial port (in a similar fashion to using Hyperterm program in Windows™).

CONCLUSION

This paper presented some of the methodology and ideology behind using Java, the JNI and the C programming language to expose students to the rigors of basic device driver development. As well, simple examples of how the JNI can be used were included to aid the reader's understanding of the simplicity of the integration of the high level programming in Java and the low-level programming in C that allows device drivers to be built and tested in a laboratory environment.

Copyright © 2007, American Society for Engineering Education"

APPENDIX – SAMPLE COURSE SCHEDULE

Week 1 Course Introduction/Introduction to Object Oriented Programming (OOP)/Introduction to Java
Laboratory 1: "Hello Java" - Tutorial

Week 2 Classes / Objects / Methods / Constructors/Introduction to GUI: Swing Components
Laboratory 2: Making Objects & Calling Methods

Week 3 Java Swing
Laboratory 3: Dialog Boxes

Week 4 Control Statements & Loops/Event Handling/Layout Managers/Panels
Laboratory 4: Designing a GUI (Week 1)

Week 5 Arrays/Exception HandlingJava I/O/Files & Streams
Laboratory 4: Designing a GUI (Week 2)

Week 6 Exam/Networking
Laboratory 5: Client-Server Chat (Week 1)

Week 7 Multi-Threading/ Fundamental 'C' Programming/Device Driver Development Overview
Laboratory 5: Client-Server Chat (Week 2)

Week 8 Fundamental 'C' Programming/Device Driver Development – the JNI
Laboratory 6: Using the JNI

Week 9 Device Driver Development – Parallel Port
Laboratory 7: LCD Device Driver for the Parallel Port (Week 1)

Week 10 Fundamental 'C' /Device Driver Development – Serial Port
Laboratory 7: LCD Device Driver for the Parallel Port (Week 2)

Week 12 'C' Programming/Device Driver Development
Laboratory 8: Serial Port Chat (Week 1)

Week 13 'C' Programming /Data Acquisition/Linux & USB
Laboratory 8: Serial Port Chat (Week 2)

Week 14 Data Representation using XML/Linux & USB
Laboratory: Lab Exam & Lab 9 Pre-lab Due

Week 15 'C' Programming /USB
Laboratory 9: Data Acquisition with the USB port (Week 1)

Week 16 Misc. Topics/Questions/Review
Laboratory 9: Data Acquisition with the USB port (Week 2)

"Proceedings of the Spring 2007 American Society for Engineering Education Illinois-Indiana Section Conference. Copyright © 2007, American Society for Engineering Education"

Bibliography

1. Barbara Johnston, *Java Programming Today,* Pearson Prentice Hall, Upper Saddle River, NJ., 2004
2. Deitel & Deitel, *Java – How to Program, Sixth edition,* Pearson Prentice Hall, Upper Saddle River, NJ.,2005.
3. Michael Waite and Stephen Prata, *New C Primer Plus,* Howard W. Sams & Company, Carmel, IN., 1990.
4. Sheng Liang, *The Java Native Interface Programmer's Guide and Specification,* Addison-Wesley, Palo Alto, CA., 1990**.**
5. Michael R. Sweet, *Serial Programming Guide for POSIX Operating System, Fifth Edition – 4th Revision,* GNU Free Documentation, 2004
6. Rich Pfile & Bill Lin, *Native Instrumentation Board Interface for Java-based Programs.* Proceedings of the 2003 ASEE Conference, 2003.

JAMES K. BROWN is an assistant clinical professor of Electrical Engineering Technology at IUPUI. He received his B.S. in Electrical Engineering from IUPUI and his M.S. Eng. from the University of South Florida. He teaches courses in device drivers, analog electronics, ethics, and senior design courses, and develops embedded and PC-based software for industry. He has three years of teaching experience and sixteen years of industrial experience.

"Proceedings of the Spring 2007 American Society for Engineering Education Illinois-Indiana Section Conference. Copyright © 2007, American Society for Engineering Education"