11

Programming Serial and Parallel Ports

11.0. Introduction

Peripheral devices are usually external to the computer.[*] Printers, mice, video cameras, scanners, data/fax modems, plotters, robots, telephones, light switches, weather gauges, Palm Computing Platform devices, and many others exist “out there,” beyond the confines of your desktop or server machine. We need a way to reach out to them.

The Java Communications API not only gives us that, but cleverly unifies the programming model for dealing with a range of external devices. It supports both serial (RS232/434, COM, or tty) and parallel (printer, LPT) ports. We’ll cover this in more detail later, but briefly, serial ports are used for modems and occasionally printers, and parallel ports are used for printers and sometimes (in the PC world) for Zip drives and other peripherals. Before USB (Universal Serial Bus) came along, it seemed that parallel ports would dominate for such peripherals, as manufacturers were starting to make video cameras, scanners, and the like. Now, however, USB has become the main attachment mode for such devices. One can imagine that future releases of Java Communications might expand the structure to include USB support (Sun has admitted that this is a possibility) and maybe other bus-like devices.

This chapter[†] aims to teach you the principles of controlling these many kinds of devices in a machine-independent way using the Java Communications API, which is in package `javax.comm`.

* Conveniently ignoring things like “internal modem cards” on desktop machines!

† This chapter was originally going to be a book. Ironic, since my first book for O’Reilly was originally going to be a chapter. So it goes.

I'll start this chapter by showing you how to get a list of available ports and how to control simple serial devices like modems. Such details as baud rate, parity, and word size are attended to before we can write commands to the modem, read the results, and establish communications. We'll move on to parallel (printer) ports, and then look at how to transfer data synchronously (using read/write calls directly) and asynchronously (using Java listeners). Then we build a simple phone dialer that can call a friend's voice phone for you—a simple phone controller, if you will. The discussion ends with a serial-port printer/plotter driver.

The Communications API

The Communications API is centered around the abstract class `CommPort` and its two subclasses, `SerialPort` and `ParallelPort`, which describe the two main types of ports found on desktop computers. `CommPort` represents a general model of communications, and has general methods like `getInputStream()` and `getOutputStream()` that allow you to use the information from Chapter 9 to communicate with the device on that port.

However, the constructors for these classes are intentionally non-public. Rather than constructing them, you instead use the static factory method `CommPortIdentifier.getPortIdentifiers()` to get a list of ports, let the user choose a port from this list, and call this `CommPortIdentifier`'s `open()` method to receive a `CommPort` object. You cast the `CommPort` object to a non-abstract subclass representing a particular communications device. At present, the subclass must be either `SerialPort` or `ParallelPort`.

Each of these subclasses has some methods that apply only to that type. For example, the `SerialPort` class has a method to set baud rate, parity, and the like, while the `ParallelPort` class has methods for setting the "port mode" to original PC mode, bidirectional mode, etc.

Both subclasses also have methods that allow you to use the standard Java event model to receive notification of events such as data available for reading, output buffer empty, and type-specific events such as ring indicator for a serial port and out-of-paper for a parallel port—as we'll see, the parallel ports were originally for printers, and still use their terminology in a few places.

About the Code Examples in This Chapter

Java Communication is a standard extension. This means that it is not a required part of the Java API, which in turn means that your vendor probably didn't ship it. You may need to download the Java Communications API from Sun's Java web site, *http://java.sun.com*, or from your system vendor's web site, and install it. If your platform or vendor doesn't ship it, you may need to find, modify, compile, and install some C code. Try my personal web site, too. And, naturally enough, to run some of the examples you will need additional peripheral devices beyond those normally provided with a desktop computer. Batteries—and peripheral devices—are not included in the purchase of this book.

See Also

Elliotte Rusty Harold's book *Java I/O* contains a chapter that discusses the Communications API in considerable detail, as well as some background issues such as baud rate that we take for granted here. Rusty also discusses some details that I have glossed over, such as the ability to set receive timeouts and buffer sizes.

This book is about portable Java. If you want the gory low-level details of setting device registers on a 16451 UART on an ISA or PCI PC, you'll have to look elsewhere; there are several books on these topics. If you really need the hardware details for I/O ports on other platforms such as Sun Workstations and Palm Computing Platform, consult either the vendor's documentation and/or the available open source operating systems that run on that platform.

11.1. Choosing a Port

Problem

You need to know what ports are available on a given computer.

Solution

Use `CommPortIdentifier.getPortIdentifiers()` to return the list of ports.

Discussion

There are many kinds of computers out there. It's unlikely that you'd find yourself running on a desktop computer with no serial ports, but you might find that there is only one and it's already in use by another program. Or you might want a parallel port and find that the computer has only serial ports. This program shows you how to use the static `CommPortIdentifier` method `getPortIdentifiers()`. This gives you an `Enumeration` (Recipe 7.4) of the serial and parallel ports available on your system. My routine `populate()` processes this list and loads it into a pair of `JComboBoxes` (graphical choosers; see Recipe 13.1), one for serial ports and one for parallel (there is also a third, unknown, to cover future expansion of the API). The routine `makeGUI` creates the `JComboBoxes` and arranges to notify us when the user picks one from either of the lists. The name of the selected port is displayed at the bottom of the window. So that you won't have to know much about it to use it, there are public methods `getSelectedName( )`, which returns the name of the last port chosen by either `JComboBox` and `getSelectedIdentifier()`, which returns an object called a `CommPortIdentifier` corresponding to the selected port name. Figure 11-1 shows the port chooser in action.

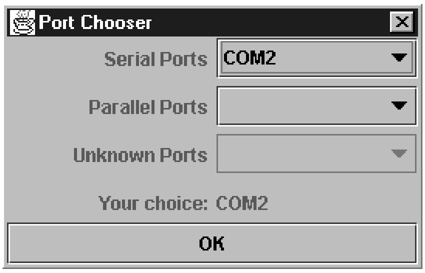


Figure 11-1. The Communications Port Chooser in action

Example 11-1 shows the code.

Example 11-1. PortChooser.java

```
import java.io.*;
import javax.comm.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.*;

/**
 * Choose a port, any port!
 *
 * Java Communications is a "standard extension" and must be downloaded
 * and installed separately from the JDK before you can even compile this
 * program.
 *
 */
public class PortChooser extends JDialog implements ItemListener {
    /** A mapping from names to CommPortIdentifiers. */
    protected HashMap map = new HashMap();
    /** The name of the choice the user made. */
    protected String selectedPortName;
    /** The CommPortIdentifier the user chose. */
    protected CommPortIdentifier selectedPortIdentifier;
    /** The JComboBox for serial ports */
    protected JComboBox serialPortsChoice;
    /** The JComboBox for parallel ports */
    protected JComboBox parallelPortsChoice;
    /** The JComboBox for anything else */
```

Example 11-1. PortChooser.java (continued)

```
    protected JComboBox other;
    /** The SerialPort object */
    protected SerialPort ttya;
    /** To display the chosen */
    protected JLabel choice;
    /** Padding in the GUI */
    protected final int PAD = 5;

    /** This will be called from either of the JComboBoxes when the
     * user selects any given item.
     */
    public void itemStateChanged(ItemEvent e) {
        // Get the name
        selectedPortName = (String)((JComboBox)e.getSource()).getSelectedItem();
        // Get the given CommPortIdentifier
        selectedPortIdentifier = (CommPortIdentifier)map.get(selectedPortName);
        // Display the name.
        choice.setText(selectedPortName);
    }

    /* The public "getter" to retrieve the chosen port by name. */
    public String getSelectedName() {
        return selectedPortName;
    }

    /* The public "getter" to retrieve the selection by CommPortIdentifier. */
    public CommPortIdentifier getSelectedIdentifier() {
        return selectedPortIdentifier;
    }

    /** A test program to show up this chooser. */
    public static void main(String[] ap) {
        PortChooser c = new PortChooser(null);
        c.setVisible(true);// blocking wait
        System.out.println("You chose " + c.getSelectedName() +
            " (known by " + c.getSelectedIdentifier() + ").");
        System.exit(0);
    }

    /** Construct a PortChooser --make the GUI and populate the ComboBoxes.
     */
    public PortChooser(JFrame parent) {
        super(parent, "Port Chooser", true);

        makeGUI();
        populate();
        finishGUI();
    }
```

Example 11-1. PortChooser.java (continued)

```
    /** Build the GUI. You can ignore this for now if you have not
     * yet worked through the GUI chapter. Your mileage may vary.
     */
    protected void makeGUI() {
        Container cp = getContentPane();

        JPanel centerPanel = new JPanel();
        cp.add(BorderLayout.CENTER, centerPanel);

        centerPanel.setLayout(new GridLayout(0,2, PAD, PAD));

        centerPanel.add(new JLabel("Serial Ports", JLabel.RIGHT));
        serialPortsChoice = new JComboBox();
        centerPanel.add(serialPortsChoice);
        serialPortsChoice.setEnabled(false);

        centerPanel.add(new JLabel("Parallel Ports", JLabel.RIGHT));
        parallelPortsChoice = new JComboBox();
        centerPanel.add(parallelPortsChoice);
        parallelPortsChoice.setEnabled(false);

        centerPanel.add(new JLabel("Unknown Ports", JLabel.RIGHT));
        other = new JComboBox();
        centerPanel.add(other);
        other.setEnabled(false);

        centerPanel.add(new JLabel("Your choice:", JLabel.RIGHT));
        centerPanel.add(choice = new JLabel());

        JButton okButton;
        cp.add(BorderLayout.SOUTH, okButton = new JButton("OK"));
        okButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                PortChooser.this.dispose();
            }
        });

    }

    /** Populate the ComboBoxes by asking the Java Communications API
     * what ports it has.  Since the initial information comes from
     * a Properties file, it may not exactly reflect your hardware.
     */
    protected void populate() {
        // get list of ports available on this particular computer,
        // by calling static method in CommPortIdentifier.
        Enumeration pList = CommPortIdentifier.getPortIdentifiers();
```

Example 11-1. PortChooser.java (continued)

```
        // Process the list, putting serial and parallel into ComboBoxes
        while (pList.hasMoreElements()) {
            CommPortIdentifier cpi = (CommPortIdentifier)pList.nextElement();
            // System.out.println("Port " + cpi.getName());
            map.put(cpi.getName(), cpi);
            if (cpi.getPortType() == CommPortIdentifier.PORT_SERIAL) {
                serialPortsChoice.setEnabled(true);
                serialPortsChoice.addItem(cpi.getName());
            } else if (cpi.getPortType() == CommPortIdentifier.PORT_PARALLEL) {
                parallelPortsChoice.setEnabled(true);
                parallelPortsChoice.addItem(cpi.getName());
            } else {
                other.setEnabled(true);
                other.addItem(cpi.getName());
            }
        }
        serialPortsChoice.setSelectedIndex(-1);
        parallelPortsChoice.setSelectedIndex(-1);
    }

    protected void finishGUI() {
        serialPortsChoice.addItemListener(this);
        parallelPortsChoice.addItemListener(this);
        other.addItemListener(this);
        pack();
        addWindowListener(new WindowCloser(this, true));
    }
}
```

11.2. Opening a Serial Port

Problem

You want to set up a serial port and open it for input/output.

Solution

Use a `CommPortIdentifier`'s `open()` method to get a `SerialPort` object.

Discussion

Now you've picked your serial port, but it's not ready to go yet. Baud rate. Parity. Stop bits. These things have been the bane of many a programmer's life. Having needed to work out the details of setting them on many platforms over the years, including CP/M systems, IBM PCs, and IBM System/370 mainframes, I can report that it's no fun. Finally, Java has provided a portable interface for setting all these parameters.

The steps in setting up and opening a serial port are as follows:

1. Get the name and `CommPortIdentifier` (which you can do using my `PortChooser` class).
2. Call the `CommPortIdentifier`'s `open()` method; cast the resulting `CommPort` object to a `SerialPort` object (this cast will fail if the user chose a parallel port!).
3. Set the serial communications parameters, such as baud rate, parity, stop bits, and the like, either individually or all at once using the convenience routing `setSerialPortParams()`.
4. Call the `getInputStream` and `getOutputStream` methods of the `SerialPort` object, and construct any additional `Stream` or `Writer` objects (see Chapter 9).

You are then ready to read and write on the serial port. Example 11-2 is code that implements all these steps for a serial port. Some of this code is for parallel ports, which we'll discuss in Recipe 11.3.

Example 11-2. CommPortOpen.java

```
import java.awt.*;
import java.io.*;
import javax.comm.*;
import java.util.*;

/**
 * Open a serial port using Java Communications.
 *
 */
public class CommPortOpen {
    /** How long to wait for the open to finish up. */
    public static final int TIMEOUTSECONDS = 30;
    /** The baud rate to use. */
    public static final int BAUD = 9600;
    /** The parent Frame, for the chooser. */
    protected Frame parent;
    /** The input stream */
    protected DataInputStream is;
    /** The output stream */
    protected PrintStream os;
    /** The last line read from the serial port. */
    protected String response;
    /** A flag to control debugging output. */
    protected boolean debug = true;
```

Example 11-2. CommPortOpen.java (continued)

```
    /** The chosen Port Identifier */
    CommPortIdentifier thePortID;
    /** The chosen Port itself */
    CommPort thePort;

    public static void main(String[] argv)
        throws IOException, NoSuchPortException, PortInUseException,
            UnsupportedCommOperationException {

        new CommPortOpen(null).converse();

        System.exit(0);
    }

    /* Constructor */
    public CommPortOpen(Frame f)
        throws IOException, NoSuchPortException, PortInUseException,
            UnsupportedCommOperationException {

        // Use the PortChooser from before. Pop up the JDialog.
        PortChooser chooser = new PortChooser(null);

        String portName = null;
        do {
            chooser.setVisible(true);

            // Dialog done. Get the port name.
            portName = chooser.getSelectedName();

            if (portName == null)
                System.out.println("No port selected. Try again.\n");
        } while (portName == null);

        // Get the CommPortIdentifier.
        thePortID = chooser.getSelectedIdentifier();

        // Now actually open the port.
        // This form of openPort takes an Application Name and a timeout.
        //
        System.out.println("Trying to open " + thePortID.getName() + "...");

        switch (thePortID.getPortType()) {
        case CommPortIdentifier.PORT_SERIAL:
            thePort = thePortID.open("DarwinSys DataComm",
                TIMEOUTSECONDS * 1000);
            SerialPort myPort = (SerialPort) thePort;
```

Example 11-2. CommPortOpen.java (continued)

```
                // set up the serial port
                myPort.setSerialPortParams(BAUD, SerialPort.DATABITS_8,
                    SerialPort.STOPBITS_1, SerialPort.PARITY_NONE);
                break;

            case CommPortIdentifier.PORT_PARALLEL:
                thePort = thePortID.open("DarwinSys Printing",
                    TIMEOUTSECONDS * 1000);
                ParallelPort pPort = (ParallelPort)thePort;

                // Tell API to pick "best available mode" - can fail!
                // myPort.setMode(ParallelPort.LPT_MODE_ANY);

                // Print what the mode is
                int mode = pPort.getMode();
                switch (mode) {
                    case ParallelPort.LPT_MODE_ECP:
                        System.out.println("Mode is: ECP");
                        break;
                    case ParallelPort.LPT_MODE_EPP:
                        System.out.println("Mode is: EPP");
                        break;
                    case ParallelPort.LPT_MODE_NIBBLE:
                        System.out.println("Mode is: Nibble Mode.");
                        break;
                    case ParallelPort.LPT_MODE_PS2:
                        System.out.println("Mode is: Byte mode.");
                        break;
                    case ParallelPort.LPT_MODE_SPP:
                        System.out.println("Mode is: Compatibility mode.");
                        break;
                    // ParallelPort.LPT_MODE_ANY is a "set only" mode;
                    // tells the API to pick "best mode"; will report the
                    // actual mode it selected.
                    default:
                        throw new IllegalStateException
                            ("Parallel mode " + mode + " invalid.");
                }
                break;
            default:// Neither parallel nor serial??
                throw new IllegalStateException("Unknown port type " + thePortID);
            }

            // Get the input and output streams
            // Printers can be write-only
            try {
                is = new DataInputStream(thePort.getInputStream());
```

Example 11-2. CommPortOpen.java (continued)

```
        } catch (IOException e) {
            System.err.println("Can't open input stream: write-only");
            is = null;
        }
        os = new PrintStream(thePort.getOutputStream(), true);
    }

    /** This method will be overridden by non-trivial subclasses
     * to hold a conversation.
     */
    protected void converse() throws IOException {

        System.out.println("Ready to read and write port.");

        // Input/Output code not written -- must subclass.

        // Finally, clean up.
        if (is != null)
            is.close();
        os.close();
    }
}
```

As noted in the comments, this class contains a dummy version of the `converse` method. In following sections we'll expand on the input/output processing by subclassing and overriding this method.

11.3. Opening a Parallel Port

Problem

You want to open a parallel port.

Solution

Use a `CommPortIdentifier`'s `open()` method to get a `ParallelPort` object.

Discussion

Enough of serial ports! Parallel ports as we know 'em are an outgrowth of the "dot matrix" printer industry. Before the IBM PC, Tandy and other "pre-PC" PC makers needed a way to hook printers to their computers. Centronics, a company that made a variety of dot matrix printers, had a standard connector mechanism that caught on, changing only when IBM got into the act. Along the way, PC makers found they needed more speed, so they built faster printer ports. And peripheral makers took advantage of this by using the faster (and by now bidirectional) printer ports to hook up all manner of weird devices like scanners, SCSI and Ethernet controllers, and others via parallel ports. You can, in theory, open any of these devices and control them; the logic of controlling such devices is left as an exercise for the reader. For now we'll just open a parallel port.

Just as the `SerialPortOpen` program set the port's parameters, the `ParallelPortOpen` program sets the parallel port access type or "mode." Like baud rate and parity, this requires some knowledge of the particular desktop computer's hardware. There are several common modes, or types of printer interface and interaction. The oldest is "simple parallel port," which the API calls MODE_SPP. This is an output-only parallel port. Other common modes include EPP (extended parallel port, MODE_ECP) and ECP (extended communciation port, MODE_ECP). The API defines a few rare ones, as well as MODE_ANY, the default, and allows the API to pick the best mode. In my experience, the API doesn't always do a very good job of picking, either with MODE_ANY or with explicit settings. And indeed, there may be interactions with the BIOS (at least on a PC) and on device drivers (MS-Windows, Unix). What follows is a simple example that opens a parallel port (though it works on a serial port also), opens a file, and sends it; in other words, a very trivial printer driver. Now this is obviously *not* the way to drive printers. Most operating systems provide support for various types of printers (the MacOS and MS-Windows both do, at least; Unix tends to assume a PostScript or HP printer). This example, just to make life simple by allowing us to work with ASCII files, copies a short file of PostScript. The intent of the PostScript job is just to print the little logo in Figure 11-2.

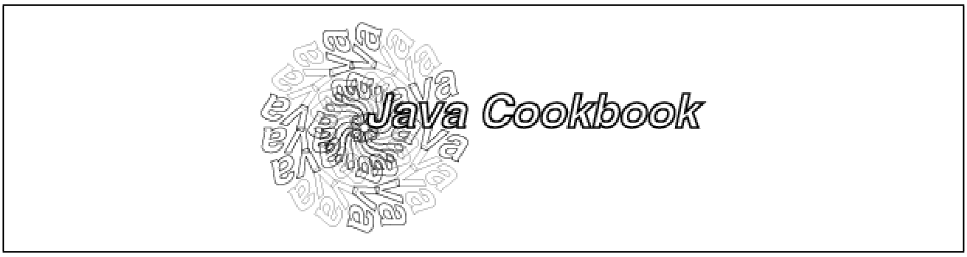


Figure 11-2. PostScript printer output

The PostScript code used in this particular example is fairly short:

```
%!PS-Adobe

% Draw a circle of "Java Cookbook"
% simplified from Chapter 9 of the Adobe Systems "Blue Book",
% PostScript Language Tutorial and Cookbook
% center the origin
250 350 translate

/Helvetica-BoldOblique findfont
    30 scalefont
    setfont

% print circle of Java
0.4 setlinewidth% make outlines not too heavy
20 20 340 {
    gsave
    rotate 0 0 moveto
    (Java) true charpath stroke
    grestore
} for

% print "Java Cookbook" in darker outline
% fill w/ light gray to contrast w/ spiral
1.5 setlinewidth
0 0 moveto
(Java Cookbook) true charpath
gsave 1 setgray fill grestore
stroke

% now send it all to the printed page
showpage
```

It doesn't matter if you know PostScript; it's just the printer control language that some printers accept. What matters to us is that we can open the parallel port, and, if an appropriate printer is connected (I used an HP6MP, which supports PostScript), the logo will print, appearing near the middle of the page. Example 11-3 is a short program that again subclasses `CommPortOpen`, opens a file that is named on the command line, and copies it to the given port. Using it looks like this:

```
C:\javasrc\io\javacomm>java ParallelPrint javacook.ps
Mode is: Compatibility mode.
Can't open input stream: write-only

C:\javasrc\io\javacomm>
```

The message "Can't open input stream" appears because my notebook's printer port is (according to the Java Comm API) unable to do bidirectional I/O. This is in fact incorrect, as I have used various printer-port devices that require bidirectional I/O, such as the Logitech (formerly Connectix) QuickCam, on this same hardware platform (but under Unix and MS-Windows, not using Java). This message is just a warning; the program works correctly despite it.

Example 11-3. ParallePrint.com

```
import java.awt.*;
import java.io.*;
import javax.comm.*;

/**
 * Print to a serial port using Java Communications.
 *
  */
public class ParallelPrint extends CommPortOpen {

    protected static String inputFileName;

    public static void main(String[] argv)
        throws IOException, NoSuchPortException, PortInUseException,
            UnsupportedCommOperationException {

        if (argv.length != 1) {
            System.err.println("Usage: ParallelPrint filename");
            System.exit(1);
        }
        inputFileName = argv[0];

        new ParallelPrint(null).converse();

        System.exit(0);
    }

    /* Constructor */
    public ParallelPrint(Frame f)
        throws IOException, NoSuchPortException, PortInUseException,
            UnsupportedCommOperationException {

        super(f);
    }

    /**
     * Hold the (one-way) conversation.
     */
    protected void converse() throws IOException {

        // Make a reader for the input file.
        BufferedReader file = new BufferedReader(
            new FileReader(inputFileName));

        String line;
        while ((line = file.readLine()) != null)
            os.println(line);
```

Example 11-3. ParallePrint.com (continued)

```
        // Finally, clean up.
        file.close();
        os.close();
    }
}
```

11.4. Resolving Port Conflicts

Problem

Somebody else is using the port you want, and they won't let go!

Solution

Use a `PortOwnershipListener`.

Discussion

If you run the `CommPortOpen` program and select a port that is opened by a native program such as HyperTerminal on MS-Windows, you will get a `PortInUseException` after the timeout period is up:

```
C:\javasrc\commport>java CommPortOpen
Exception in thread "main" javax.comm.PortInUseException: Port currently owned by
Unknown Windows Application
        at javax.comm.CommPortIdentifier.open(CommPortIdentifier.java:337)
        at CommPortOpen.main(CommPortOpen.java:41)
```

If, on the other hand, you run two copies of `CommPortOpen` at the same time for the same port, you will see something like the following:

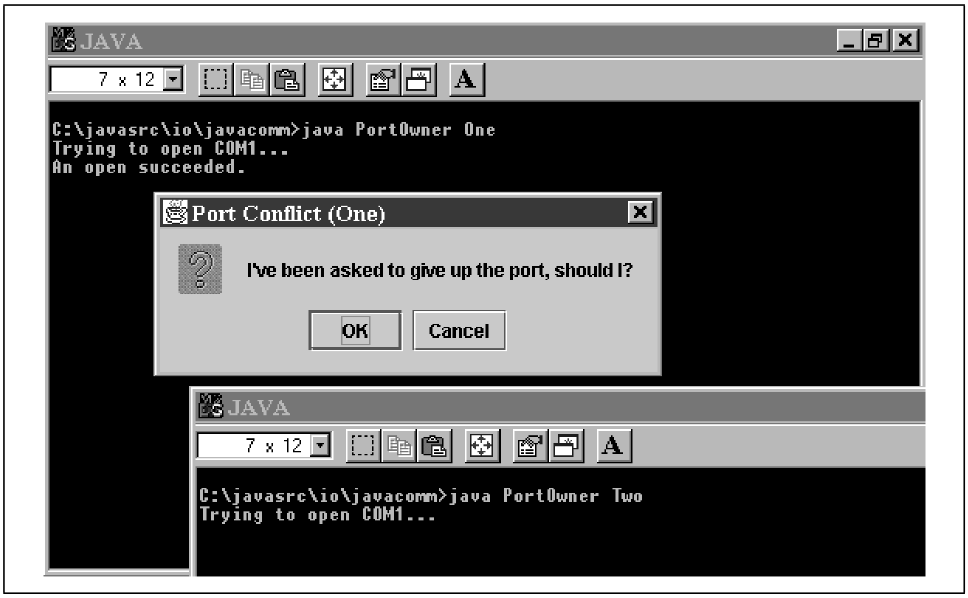
```
C:\javasrc\commport>java CommPortOpen
Exception in thread "main" javax.comm.PortInUseException: Port currently owned by
DarwinSys DataComm
        at javax.comm.CommPortIdentifier.open(CommPortIdentifier.java:337)
        at CommPortOpen.main(CommPortOpen.java:41)

C:\javasrc\commport>
```

To resolve conflicts over port ownership, you can register a `PortOwnershipListener` so that you will be told if another application wants to use the port. Then you can either close the port and the other application will get it, or ignore the request and the other program will get a `PortInUseException`, as we did here.

What is this “listener”? The Event Listener model is used in many places in Java. It may be best known for its uses in GUIs (see Recipe 13.4). The basic form is that you have to *register* an object as a *listener* with an *event source*. The event source will then call a well-known method to notify you that a particular event has occurred. In the GUI, for example, an event occurs when the user presses a button with the mouse; if you wish to monitor these events, you need to call the button object’s `addActionListener()` method, passing an instance of the `ActionListener` interface (which can be your main class, an inner class, or some other class).

How does a listener work in practice? To simplify matters, we’ve again subclassed from our command-line program `CommPortOpen` to pop up a dialog if one copy of the program tries to open a port that another copy already has open. If you run two copies of the new program `PortOwner` at the same time, and select the same port in each, you’ll see the dialog shown in Figure 11-3.



```
C:\javasrc\io\javacomm>java PortOwner One
Trying to open COM1...
An open succeeded.
```

I've been asked to give up the port, should I?

```
C:\javasrc\io\javacomm>java PortOwner Two
Trying to open COM1...
```

Figure 11-3. Port conflict resolution

The trick to make this happen is simply to add a `CommPortOwnershipListener` to the `CommPortIdentifier` object. You will then be called when any program gets ownership, gives up ownership, or if there is a conflict. Example 11-4 shows the program with this addition.

Example 11-4. PortOwner.java

```
import javax.comm.*;
import java.io.*;
import javax.swing.*;

/** Demonstrate the port conflict resolution mechanism.
 * Run two copies of this program and choose the same port in each.
 */
public class PortOwner extends CommPortOpen {
    /** A name for showing which of several instances of this program */
    String myName;

    public PortOwner(String name)
        throws IOException, NoSuchPortException, PortInUseException,
            UnsupportedCommOperationException {

        super(null);
        myName = name;
        thePortID.addPortOwnershipListener(new MyResolver());
    }

    public void converse() {
        // lah de dah...
        // To simulate a long conversation on the port...

        try {
            Thread.sleep(1000 * 1000);
        } catch (InterruptedException cantHappen) {
            //
        }
    }

    /** An inner class that handles the ports conflict resolution. */
    class MyResolver implements CommPortOwnershipListener {
        protected boolean owned = false;
        public void ownershipChange(int whaHoppen) {
            switch (whaHoppen) {
            case PORT_OWNED:
                System.out.println("An open succeeded.");
                owned = true;
                break;
            case PORT_UNOWNED:
                System.out.println("A close succeeded.");
                owned = false;
                break;
            case PORT_OWNERSHIP_REQUESTED:
                if (owned) {
                    if (JOptionPane.showConfirmDialog(null,
                        "I've been asked to give up the port, should I?",
```

Example 11-4. PortOwner.java (continued)

```
                        "Port Conflict (" + myName + ")",
                        JOptionPane.OK_CANCEL_OPTION) == 0)
                    thePort.close();
                } else {
                    System.out.println("Somebody else has the port");
                }
            }
        }
    }

    public static void main(String[] argv)
        throws IOException, NoSuchPortException, PortInUseException,
            UnsupportedCommOperationException {

        if (argv.length != 1) {
            System.err.println("Usage: PortOwner aname");
            System.exit(1);
        }

        new PortOwner(argv[0]).converse();

        System.exit(0);
    }
}
```

Note the single argument to `ownershipChange()`. Do not assume that only your listener will be told when an event occurs; it will be called whether you are the affected program or simply a bystander. To see if you are the program being requested to give up ownership, you have to check to see if you already have the port that is being requested (for example, by opening it successfully!).

11.5. Reading and Writing: Lock Step

Problem

You want to read and write on a port, and your communications needs are simple.

Solution

Just use read and write calls.

Discussion

Suppose you need to send a command to a device and get a response back, and then send another, and get another. This has been called a “lock-step” protocol, since both ends of the communication are locked into step with one another, like soldiers on parade. There is no requirement that both ends be able to write at the same time (see Recipes 10.7 and 10.8 for this), since you know what the response to your command should be and don't proceed until you have received that response. A well-known example is using a standard Hayes-command-set modem to just dial a phone number. In its simplest form, you send the command string ATZ and expect the response OK, then send ATD with the number, and expect CONNECT. To implement this, we first subclass from `CommPortOpen` to add two functions, `send` and `expect`, which perform reasonably obvious functions for dealing with such devices. See Example 11-5.

Example 11-5. CommPortModem.java

```
import java.awt.*;
import java.io.*;
import javax.comm.*;
import java.util.*;

/**
 * Subclasses CommPortOpen and adds send/expect handling for dealing
 * with Hayes-type modems.
 *
 */
public class CommPortModem extends CommPortOpen {
    /** The last line read from the serial port. */
    protected String response;
    /** A flag to control debugging output. */
    protected boolean debug = true;

    public CommPortModem(Frame f)
        throws IOException, NoSuchPortException,PortInUseException,
            UnsupportedCommOperationException {
        super(f);
    }

    /** Send a line to a PC-style modem. Send \r\n, regardless of
     * what platform we're on, instead of using println().
     */
    protected void send(String s) throws IOException {
        if (debug) {
            System.out.print(">>> ");
            System.out.print(s);
            System.out.println();
        }
        os.print(s);
        os.print("\r\n");

        // Expect the modem to echo the command.
```

Example 11-5. CommPortModem.java (continued)

```
        if (!expect(s)) {
            System.err.println("WARNING: Modem did not echo command.");
        }

        // The modem sends an extra blank line by way of a prompt.
        // Here we read and discard it.
        String junk = os.readLine();
        if (junk.length() != 0) {
            System.err.print("Warning unexpected response: ");
            System.err.println(junk);
        }
    }

    /** Read a line, saving it in "response".
     * @return true if the expected String is contained in the response, false if not.
     */
    protected boolean expect(String exp) throws IOException {
        response = is.readLine();
        if (debug) {
            System.out.print("<<< ");
            System.out.print(response);
            System.out.println();
        }
        return response.indexOf(exp) >= 0;
    }
}
```

Finally, Example 11-6 extends our CommPortModem program to initialize the modem and dial a telephone number.

Example 11-6. CommPortDial.java

```
import java.io.*;
import javax.comm.*;
import java.util.*;

/**
 * Dial a phone using the Java Communications Package.
 *
 */
public class CommPortDial extends CommPortModem {

    protected static String number = "000-0000";

    public static void main(String[] ap)
        throws IOException, NoSuchPortException,PortInUseException,
            UnsupportedCommOperationException {
        if (ap.length == 1)
```

Example 11-6. CommPortDial.java (continued)

```
            number = ap[0];
        new CommPortDial().converse();
        System.exit(0);
    }

    public CommPortDial()
        throws IOException, NoSuchPortException, PortInUseException,
            UnsupportedCommOperationException {
        super(null);
    }

    protected void converse() throws IOException {

        String resp;// the modem response.

        // Send the reset command
        send("ATZ");

        expect("OK");

        send("ATDT" + number);

        expect("OK");

        try {
            Thread.sleep(5000);
        } catch (InterruptedException e) {
            // nothing to do
        }
        is.close();
        os.close();
    }
}
```

11.6. Reading and Writing: Event-Driven

Problem

After the connection is made, you don't know what order to read or write in.

Solution

Use Java Communication Events to notify you when data becomes available.

Discussion

While lock-step mode is acceptable for dialing a modem, it breaks down when you have two independent agents communicating over a port. Either end may be a person, as in a remote login session, or a program, either a server or a client program. A client program, in turn, may be driven by a person (as is a web browser) or may be self-driven (such as an FTP client transferring many files at one request). You cannot predict, then, who will need to read and who will need to write. Consider the simplest case: the programs at both end try to read at the same time! Using the lock-step model, each end will wait forever for the other end to write something. This error condition is known as a *deadlock*, since both ends are locked up, dead, until a person intervenes, or the communication line drops, or the world ends, or the universe ends, or somebody making tea blows a fuse and causes one of the machines to halt.

There are two general approaches to this problem: event-driven activity, wherein the Communications API notifies you when the port is ready to be read or written, and threads-based activity, wherein each "direction" (from the user to the remote, and from the remote to the user) has its own little flow of control, causing only the reads in that direction to wait. We'll discuss each of these.

First, Example 11-7 reads from a serial port using the event-driven approach.

Example 11-7. SerialReadByEvents.java

```
import java.awt.*;
import java.io.*;
import javax.comm.*;
import java.util.*;

/**
 * Read from a Serial port, notifying when data arrives.
 * Simulation of part of an event-logging service.
 */
public class SerialReadByEvents extends CommPortOpen
    implements SerialPortEventListener {

    public static void main(String[] argv)
        throws IOException, NoSuchPortException, PortInUseException,
            UnsupportedCommOperationException {

        new SerialReadByEvents(null).converse();
    }

    /* Constructor */
    public SerialReadByEvents(Frame f)
        throws IOException, NoSuchPortException, PortInUseException,
            UnsupportedCommOperationException {
```

Example 11-7. SerialReadByEvents.java (continued)

```
        super(f);
    }

    protected BufferedReader ifile;

    /**
     * Hold the conversation.
     */
    protected void converse() throws IOException {

        if (!(thePort instanceof SerialPort)) {
            System.err.println("But I wanted a SERIAL port!");
            System.exit(1);
        }
        // Tell the Comm API that we want serial events.
        ((SerialPort)thePort).notifyOnDataAvailable(true);
        try {
            ((SerialPort)thePort).addEventListener(this);
        } catch (TooManyListenersException ev) {
            // "CantHappen" error
            System.err.println("Too many listeners(!) " + ev);
            System.exit(0);
        }

        // Make a reader for the input file.
        ifile = new BufferedReader(new InputStreamReader(is));

        //
    }
    public void serialEvent(SerialPortEvent ev) {
        String line;
        try {
            line = ifile.readLine();
            if (line == null) {
                System.out.println("EOF on serial port.");
                System.exit(0);
            }
            os.println(line);
        } catch (IOException ex) {
            System.err.println("IO Error " + ex);
        }
    }
}
```

As you can see, the `serialEvent()` method does the `readLine()` calls. "But wait!" I hear you say. "This program is not a very meaningful example. It could just as easily be implemented using the lock-step method of Recipe 11.5." True enough, gentle reader. Have patience with your humble and obedient servant. Here is a program that will read from each and any of the serial ports, whenever data arrives. The program is representative of a class of programs called “data loggers,” which receive data from a number (possibly a large number) of remote locations, and log them centrally. One example is a burglar alarm monitoring station, which needs to log activities such as the alarm being turned off at the close of the day, entry by the cleaners later, what time they left, and so on. And then, of course, it needs to notify the operator of the monitoring station when an unexpected event occurs. This last step is left as an exercise for the reader.

Example 11-8 makes use of the `EventListener` model and uses a unique instance of the inner class `Logger` for each serial port it’s able to open.

Example 11-8. SerialLogger.java

```
import java.io.*;
import javax.comm.*;
import java.util.*;

/**
 * Read from multiple Serial ports, notifying when data arrives on any.
 */
public class SerialLogger {

    public static void main(String[] argv)
        throws IOException, NoSuchPortException, PortInUseException,
            UnsupportedCommOperationException {

        new SerialLogger();
    }

    /* Constructor */
    public SerialLogger()
        throws IOException, NoSuchPortException, PortInUseException,
            UnsupportedCommOperationException {

        // get list of ports available on this particular computer,
        // by calling static method in CommPortIdentifier.
        Enumeration pList = CommPortIdentifier.getPortIdentifiers();

        // Process the list, putting serial and parallel into ComboBoxes
        while (pList.hasMoreElements()) {
            CommPortIdentifier cpi = (CommPortIdentifier)pList.nextElement();
            String name = cpi.getName();
            System.out.print("Port " + name + " ");
            if (cpi.getPortType() == CommPortIdentifier.PORT_SERIAL) {
                System.out.println("is a Serial Port: " + cpi);
```

Example 11-8. SerialLogger.java (continued)

```
                SerialPort thePort;
                try {
                    thePort = (SerialPort)cpi.open("Logger", 1000);
                } catch (PortInUseException ev) {
                    System.err.println("Port in use: " + name);
                    continue;
                }

                // Tell the Comm API that we want serial events.
                thePort.notifyOnDataAvailable(true);
                try {
                    thePort.addEventListener(new Logger(cpi.getName(), thePort));
                } catch (TooManyListenersException ev) {
                    // "CantHappen" error
                    System.err.println("Too many listeners(!) " + ev);
                    System.exit(0);
                }
            }
        }
    }

    /** Handle one port. */
    public class Logger implements SerialPortEventListener {
        String portName;
        SerialPort thePort;
        BufferedReader ifile;
        public Logger(String name, SerialPort port) throws IOException {
            portName = name;
            thePort = port;
            // Make a reader for the input file.
            ifile = new BufferedReader(
                new InputStreamReader(thePort.getInputStream()));
        }
        public void serialEvent(SerialPortEvent ev) {
            String line;
            try {
                line = ifile.readLine();
                if (line == null) {
                    System.out.println("EOF on serial port.");
                    System.exit(0);
                }
                System.out.println(portName + ": " + line);
            } catch (IOException ex) {
                System.err.println("IO Error " + ex);
            }
        }
    }
}
```

11.7. Reading and Writing: Threads

Problem

After the connection is made, you don't know what order to read or write in.

Solution

Use a *thread* to handle each direction.

Discussion

When you have two things that must happen at the same time or unpredictably, the normal Java paradigm is to use a thread for each. We will discuss threads in detail in Chapter 24, but for now, just think of a thread as a small, semi-independent flow of control within a program, just as a program is a small, self-contained flow of control within an operating system. The Thread API requires you to construct a method whose signature is `public void run()` to do the body of work for the thread, and call the `start()` method of the thread to "ignite" it and start it running independently. This example creates a `Thread` subclass called `DataThread`, which reads from one file and writes to another. `DataThread` works a byte at a time so that it will work correctly with interactive prompts, which don't end at a line ending. My now-familiar `converse()` method creates two of these `DataThreads`, one to handle data "traffic" from the keyboard to the remote, and one to handle bytes arriving from the remote and copy them to the standard output. For each of these the `start()` method is called. Example 11-9 shows the entire program.

Example 11-9. CommPortThreaded.java

```
import java.io.*;
import javax.comm.*;
import java.util.*;

/**
 * This program tries to do I/O in each direction using a separate Thread.
  */
public class CommPortThreaded extends CommPortOpen {

    public static void main(String[] ap)
        throws IOException, NoSuchPortException,PortInUseException,
            UnsupportedCommOperationException {
        CommPortThreaded cp;
        try {
            cp = new CommPortThreaded();
```

Example 11-9. CommPortThreaded.java (continued)

```
            cp.converse();
        } catch(Exception e) {
            System.err.println("You lose!");
            System.err.println(e);
        }
    }

    public CommPortThreaded()
        throws IOException, NoSuchPortException, PortInUseException,
            UnsupportedCommOperationException {
        super(null);
    }

    /** This version of converse() just starts a Thread in each direction.
     */
    protected void converse() throws IOException {

        String resp;// the modem response.

        new DataThread(is, System.out).start();
        new DataThread(new DataInputStream(System.in), os).start();

    }

    /** This inner class handles one side of a conversation. */
    class DataThread extends Thread {
        DataInputStream inStream;
        PrintStream pStream;

        /** Construct this object */
        DataThread(DataInputStream is, PrintStream os) {
            inStream = is;
            pStream = os;
        }

        /** A Thread's run method does the work. */
        public void run() {
            byte ch = 0;
            try {
                while ((ch = (byte)inStream.read()) != -1)
                    pStream.print((char)ch);
            } catch (IOException e) {
                System.err.println("Input or output error: " + e);
                return;
            }
        }
    }
}
```

11.8. Program: Penman Plotter

This program in Example 11-10 is an outgrowth of the `Plotter` class from Recipe 8.11. It connects to a *Penman* plotter. These serial-port plotters were made in the United Kingdom in the 1980s, so it is unlikely that you will meet one. However, there are several companies that still make pen plotters. See Figure 11-4 for a photograph of the plotter in action.

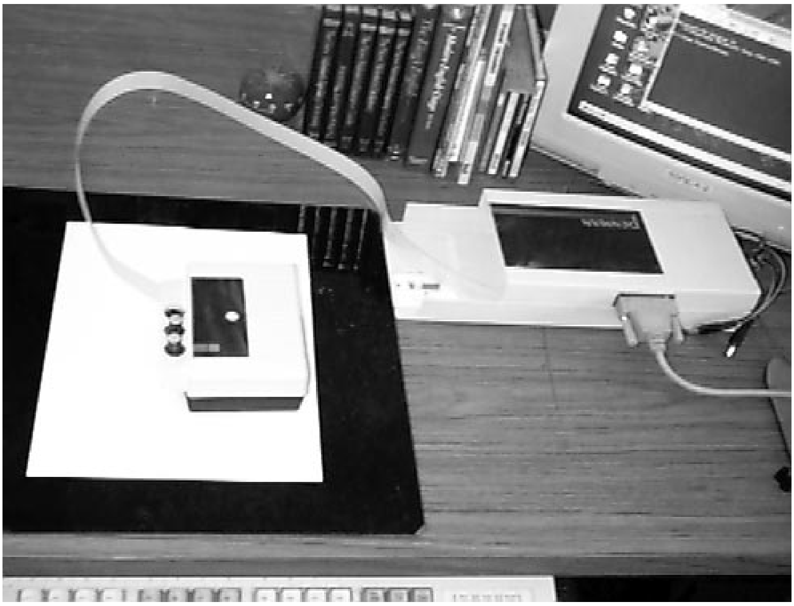


Figure 11-4. Penman plotter in action

Example 11-10. Penman.java

```java
import java.io.*;
import javax.comm.*;
import java.util.*;

/**
 * A Plotter subclass for drawing on a Penman plotter.
 * These were made in the UK and sold into North American markets.
 * It is a little "turtle" style robot plotter that communicates
 * over a serial port. For this, we use the "Java Communications" API.
 * Java Communications is a "standard extension" and must be downloaded
 * and installed separately from the JDK before you can even compile this
 * program.
 *
 */
```

Example 11-10. Penman.java (continued)

```
public class Penman extends Plotter {
    private final String OK_PROMPT = "\r\n!";
    private final int MAX_REPLY_BYTES = 50;// paranoid upper bound
    private byte b, reply[] = new byte[22];
    private SerialPort tty;
    private DataInputStream is;
    private DataOutputStream os;

    /** Construct a Penman plotter object */
    public Penman() throws NoSuchPortException,PortInUseException,
            IOException,UnsupportedCommOperationException {
        super();
        init_comm("COM2");// setup serial commx
        init_plotter();// set plotter to good state
    }

    private void init_plotter() {
        send("I"); expect('!');// eat VERSION etc., up to !
        send("I"); expect('!');// wait for it!
        send("H");// find home position
        expect('!');// wait for it!
        send("A");// Set to use absolute coordinates
        expect('!');
        curx = cury = 0;
        penUp();
    }

    //
    // PUBLIC DRAWING ROUTINES
    //

    public void setFont(String fName, int fSize) {
        // Font name is ignored for now...

        // Penman's size is in mm, fsize in points (inch/72).
        int size = (int)(fSize*25.4f/72);
        send("S"+size + ","); expect(OK_PROMPT);
        System.err.println("Font set request: " + fName + "/" + fSize);
    }

    public void drawString(String mesg) {
        send("L" + mesg + "\r"); expect(OK_PROMPT);
    }

    /** Move to a relative location */
    public void rmoveTo(int incrx, int incry){
        moveTo(curx + incrx, cury + incry);
    }
```

Example 11-10. Penman.java (continued)

```
    /** move to absolute location */
    public void moveTo(int absx, int absy) {
        System.err.println("moveTo ["+absx+","+absy+"]");
        curx = absx;
        cury = absy;
        send("M" + curx + "," + cury + ","); expect(OK_PROMPT);
    }

    private void setPenState(boolean up) {
        penIsUp = up;
        System.err.println("Pen Up is ["+penIsUp+"]");
    }

    public void penUp() {
        setPenState(true);
        send("U"); expect(OK_PROMPT);
    }
    public void penDown() {
        setPenState(false);
        send("D"); expect(OK_PROMPT);
    }
    public void penColor(int c) {
        penColor = (c%3)+1;// only has 3 pens, 4->1
        System.err.println("PenColor is ["+penColor+"]");
        send("P" + c + ","); expect(OK_PROMPT);
    }

    //
    // PRIVATE COMMUNICATION ROUTINES
    //

    private void init_comm(String portName) throws
              NoSuchPortException, PortInUseException,
             IOException, UnsupportedCommOperationException {

        // get list of ports available on this particular computer.
        // Enumeration pList = CommPortIdentifier.getPortIdentifiers();

        // Print the list. A GUI program would put these in a chooser!
        // while (pList.hasMoreElements()) {
            // CommPortIdentifier cpi = (CommPortIdentifier)pList.nextElement();
            // System.err.println("Port " + cpi.getName());
        // }

        // Open a port.
        CommPortIdentifier port =
            CommPortIdentifier.getPortIdentifier(portName);
```

Example 11-10. Penman.java (continued)

```
        // This form of openPort takes an Application Name and a timeout.
        tty = (SerialPort) port.openPort("Penman Driver", 1000);

        // set up the serial port
        tty.setSerialPortParams(9600, SerialPort.DATABITS_8,
            SerialPort.STOPBITS_1, SerialPort.PARITY_NONE);
        tty.setFlowcontrolMode(SerialPort.FLOWCTRL_RTSCTS_OUT|
            SerialPort.FLOWCTRL_RTSCTS_OUT);

        // Get the input and output streams
        is = new DataInputStream(tty.getInputStream());
        os = new DataOutputStream(tty.getOutputStream());
    }

    /** Send a command to the plotter. Although the argument is a String,
     * we send each char as a *byte*, so avoid 16-bit characters!
     * Not that it matters: the Penman only knows about 8-bit chars.
     */
    privatevoid send(String s) {
        System.err.println("sending " + s + "...");
        try {
            for (int i=0; i<s.length(); i++)
                os.writeByte(s.charAt(i));
        } catch(IOException e) {
            e.printStackTrace();
        }
    }

    /** Expect a given CHAR for a result */
    privatevoid expect(char s) {
        byte b;
        try {
            for (int i=0; i<MAX_REPLY_BYTES; i++){
                if ((b = is.readByte()) == s) {
                        return;
                }
                System.err.print((char)b);
            }
        } catch (IOException e) {
            System.err.println("Penman:expect(char "+s+"): Read failed");
            System.exit(1);
        }
        System.err.println("ARGHH!");
    }

    /** Expect a given String for a result */
    privatevoid expect(String s) {
        byte ans[] = new byte[s.length()];
```

Example 11-10. Penman.java (continued)

```
        System.err.println("expect " + s + " ...");
        try {
            is.read(ans);
        } catch (IOException e) {
            System.err.println("Penman:expect(String "+s+"): Read failed");
            System.exit(1);
        };
        for (int i=0; i<s.length() && i<ans.length; i++)
            if (ans[i] != s.charAt(i)) {
                System.err.println("MISMATCH");
                break;
            }
        System.err.println("GOT: " + new String(ans));

    }
}
```

See Also

In the online source there is a program called JModem, which implements remote connections (like *tip* or *cu* on Unix, or HyperTerminal on MS-Windows). It is usable, but too long to include in this book.

There are other specialized APIs for dealing with particular devices. For communicating with Palm Computing Platform devices, you can either use the Palm SDK for Java from Palm Computing, or the third-party API jSyncManager by Brad Barclay, which can be obtained from *http://web.idirect.com/~warp/*.