

```
/*
    Problem 7B:      DME Pane Driver
                    Stores information for multiple readings
                    in a two-dimensional array. Usage is
                    stored in a single dimension array along with
                    an appended value to show original table position
                    before the array is sorted. Fill option is used
                    to avoid having non-loaded table positions show up
                    in the final for minimum and maximum values. For table
                    processing purposes, this application uses a try with
                    multiple catch blocks for error trapping. Finally,
                    a Vector is employed to retain customer objects for
                    processing in a summary report.
    NOTE:           This application does not include inheritance, but rather
                    produces objects so they may be retained in a Vector.
    Programmer:     Wayne Spence
    Date:           November 16, 2003
    Program Name:   DME7BPaneApp
*/
```

```
import javax.swing.*;
import java.text.*;
import java.util.*;

public class DME7BPaneApp
{
    public void DME7BPaneApp()
    {
    }

    public void PaneApp()
    {
        //Declaring Variables
        int recordCount = 0;
        int kilowattHoursUsed;
        double fuelChargeRate;
        double fuelCharge;
        int numMeters;
        int j = 0;

        String outputLine = "";
        String moreInput = "yes";
    }
}
```

```

GregorianCalendar dateTodayGreg = new GregorianCalendar();
Date dateToday = new Date();
DateFormat stdDateFormat = DateFormat.getDateInstance(DateFormat.FULL);
String dateTodayString = stdDateFormat.format(dateToday);
Date meterReadingDate = null;
String meterReadingDateString = null;
int currentYear = dateTodayGreg.get(Calendar.YEAR);
int currentMonth = dateTodayGreg.get(Calendar.MONTH) + 1;
int currentDay = dateTodayGreg.get(Calendar.DAY_OF_MONTH);
Vector allData = new Vector();

while (!moreInput.equalsIgnoreCase("no"))
{
    DME7ACustomer customer = new DME7ACustomer();
    // Prompt and get input from user
    customer.setCustomerNumber();
customer.setCustomerType();
    customer.setServiceType();
    boolean validDate = false;
    do
    {
customer.setMeterReadingDay(currentDay);
customer.setMeterReadingMonth(currentMonth);
        /*
        Winter Usage (Nov thru Apr)    0.025 per kilowatt
        Summer Usage (May thru Oct)    0.030 per kilowatt
        */

        switch (customer.getMeterReadingMonth())
        {
            case 1:
            case 2:
            case 3:
            case 4: fuelChargeRate = 0.025;
                    break;

            case 5:
            case 6:
            case 7:
            case 8:
            case 9:
            case 10: fuelChargeRate = 0.030;
                    break;
            default: fuelChargeRate = 0.025;
        }
    }
}

```

```

}
customer.setMeterReadingYear(currentYear);
    GregorianCalendar meterReadingDateGreg = new
        GregorianCalendar(customer.getMeterReadingYear(), customer.getMeterReadingMonth()-1,
            customer.getMeterReadingDay());
meterReadingDate = meterReadingDateGreg.getTime();
meterReadingDateString = stdDateFormat.format(meterReadingDate);
if (meterReadingDateGreg.before(dateTodayGreg))
    validDate = true;
else
{
    outputLine = "Your transaction date of " + meterReadingDateString +
        " is a future date. \nPlease, try again";
    JOptionPane.showMessageDialog (null, outputLine);
}
}while (!validDate);
numMeters = 1;
customer.setNumberOfMeters(numMeters);
numMeters = customer.getNumberOfMeters();
String[][] outputData = new String[numMeters][5];
int[] usageStats = new int[numMeters];
Arrays.fill(usageStats, 999999);
int totalKilowattHoursUsed = 0;
NumberFormat currency = NumberFormat.getCurrencyInstance();
for (int i = 1; i<=numMeters; i++)
{
    j = i - 1;
    customer.setCurrentReading();
    if (customer.getCurrentReading() < 1)
        break;
    // Calculation fuel charge
    kilowattHoursUsed = customer.getCurrentReading() - customer.getLastReading();
    outputData[j][0] = Integer.toString(customer.getLastReading());
    outputData[j][1] = Integer.toString(customer.getCurrentReading());
    outputData[j][2] = Integer.toString(kilowattHoursUsed);
    outputData[j][3] = Double.toString(fuelChargeRate);
    outputData[j][4] = currency.format(fuelChargeRate * kilowattHoursUsed);
    usageStats[j] = kilowattHoursUsed * 10 + j;
    totalKilowattHoursUsed+= kilowattHoursUsed;
}
allData.add(customer);
fuelCharge = Math.round(totalKilowattHoursUsed * fuelChargeRate * 100.0)/100.0;

```

```

String fuelChargeString = currency.format(fuelCharge);

// Create and produce output
    recordCount++;
    String paneTitlePlus = DME7ACustomer.DME_PANE_TITLE + ": " + recordCount + " record(s)
processed.";
outputLine = "The following transaction set was recorded on: " + dateTodayString + "\n" +
    "\n Customer " + customer.getCustomerName() + " meter was last read on " + meterReadingDateString
+ "\n";
outputPreparation:
    for (int i = 1; i <= numMeters; i++)
    {
        j = i - 1;
        for (int k = 0; k <= 4; k++)
        {
            try
            {
                if (!outputData[j][k].equals(null))
                    outputLine = outputLine + "      " + outputData[j][k];
                if (Integer.parseInt(outputData[j][2]) <= 0)
                    throw new ArrayIndexOutOfBoundsException();
            }
            catch (NullPointerException e)
            {
                numMeters = j;
                j--;
                System.out.println("Exception value is " + e);
                break outputPreparation;
            }
            catch (ArrayIndexOutOfBoundsException ex)
            {
                int errorLocation = j + 1;
                outputLine = "Array process error for meter reading " + errorLocation;
                JOptionPane.showMessageDialog(null, outputLine,
                    DME7ACustomer.DME_PANE_TITLE, JOptionPane.ERROR_MESSAGE);
                System.exit(0);
            }
        }
        outputLine = outputLine + "\n";
    }
Arrays.sort(usageStats);
int low = usageStats[0] - usageStats[0]/10 * 10;

```

```

        int high = usageStats[j] - usageStats[j]/10 * 10;
        outputLine = outputLine + "\nThe fuel charge for " + totalKilowattHoursUsed +
            " total KWH used at a rate of " + fuelChargeRate + " per kilowatt hour" +
            " is " + fuelChargeString +
            "\n\nOf all meters, the lowest usage was " + outputData[low][2] +
            " kilowatts. The highest usage was " + outputData[high][2] + ".";
        JOptionPane.showMessageDialog(null, outputLine,
            paneTitlePlus, JOptionPane.PLAIN_MESSAGE);
        moreInput = JOptionPane.showInputDialog(null, "Do you wish to continue entering input? (Yes or
No)",
            DME7ACustomer.DME_PANE_TITLE, JOptionPane.PLAIN_MESSAGE);
    }
    int grandSize = allData.size();
    outputLine = "The following customers were processed during this session: \n";
    for (int i = 0; i <= grandSize -1; i++)
    {
        DME7ACustomer summaryCustomer = (DME7ACustomer) allData.get(i);
        outputLine = outputLine + "    -" + summaryCustomer.getCustomerName() + " \n";
    }
    String paneTitleSummary = DME7ACustomer.DME_PANE_TITLE + ": " + grandSize + " Customer(s) processed.";
    JOptionPane.showMessageDialog(null, outputLine,
        paneTitleSummary, JOptionPane.PLAIN_MESSAGE);
}
}

```