```
// BankcardKid.java - ATM USER EMULATOR
//
// MODULE INDEX
// NAME                          CONTENTS
// BankcardKid.randWait          Wait for a pseudo-random period of time
// BankcardKid.loadCardData      Load card data
// BankcardKid.enterString       Enter a string into the keypad
// BankcardKid.compareScreen     Compare screen with previous dump
// BankcardKid.main              Main line
//
// MAINTENANCE HISTORY
// DATE         PROGRAMMER AND DETAILS
// 19-08-07     JS      Original
//
//-----------------------------------------------------------------------------

// IMPORTS

import java.util.*;
import java.io.*;

//-----------------------------------------------------------------------------

public class BankcardKid {

    //-------------------------------------------------------------------------

    // DEFINITIONS

    static private final int INI_CARD_DATA_CAP = 60;
                                // Initial card data capacity
    static private final int CARD_DATA_CAP_INC = 10;
                                // Card data capacity increment
    static private final String CARD_DATA_FILE_NAME = "TestCards.txt";
                                // Card data file name
    static private final int PIN_LEN = 6;
                                // Length of the PIN string
    static private final int PIX_ARR_SIZE = 307200;
                                // Pixel array size
    static private final int MAX_CHG_PIX = 200;
                                // Maximum changed pixels
    static private final int MIN_INTER_KEY_WAIT = 50;
                                // Minimum inter-key wait
    static private final int MAX_INTER_KEY_WAIT = 1000;
                                // Maximum inter-key wait
    static private final int DISP_UNIT = 10;
                                // Dispensing unit (dollars)
    static private final int MAX_AMT = 1000;
                                // Maximum amount to dispense (dollars)
    static private final int MIN_AMT = 10;
                                // Minimum amount to dispense (dollars)
    static private final int CENTS_LEN = 2;
                                // Length of the cents field

    //-------------------------------------------------------------------------

    // CARD DATA STRUCTURE

    static private class CardData {
        String          cardNumber;     // Card number
        String          cardPIN;        // Card PIN
    }

    //-------------------------------------------------------------------------
```

```
// STATIC VARIABLES

static private Random randGen = new Random ();
                                        // Random number generator
static private CardData[] cardDataArr = loadCardData();
                                        // Card-PIN data array
static private long[] acpCardPix =
    ATMInfra.readScreenDump ("AcceptedCard.rgb");
                                        // Accepted card pixels
static private long[] acpPINPix =
    ATMInfra.readScreenDump ("AcceptedPIN.rgb");
                                        // Accepted PIN pixels
static private long[] acpAccPix =
    ATMInfra.readScreenDump ("AcceptedAccount.rgb");
                                        // Accepted account pixels

//-----------------------------------------------------------------------

// WAIT FOR A PSEUDO-RANDOM PERIOD OF TIME

static private void
randWait (
    int             minWait,        // Minimum waiting period (ms)
    int             maxWait)        // Maximum waiting period (ms)
{
    int             period;         // Period to sleep for

    try {
        period = minWait + randGen.nextInt (maxWait - minWait + 1);
        Thread.sleep (period);
    }
    catch (InterruptedException e) {
        throw new RuntimeException (e.toString());
    }
}

//-----------------------------------------------------------------------

// LOAD CARD DATA

static public CardData[]
loadCardData ()
{
    Vector<CardData> cardDataVec;   // Card data vector
    FileReader      fileReader;     // File reader instance
    StringBuffer    stringBuffer;   // String buffer
    int             ch;             // Received character
    CardData        cardData;       // Card data instance

    // Create the card data vector

    cardDataVec = new Vector<CardData>
        (INI_CARD_DATA_CAP, CARD_DATA_CAP_INC);

    // Create the string buffer

    stringBuffer = new StringBuffer();

    // Catch exceptions

    try {
        // Open the file and load the look-ahead character
```

```
    fileReader = new FileReader (CARD_DATA_FILE_NAME);
    ch = fileReader.read();

    // Read card data lines until end-of-file is encountered

    while (ch != -1) {
        cardData = new CardData();

        // Read the card number

        if ( ! Character.isDigit(ch))
            throw new RuntimeException ("invalid char in card data");
        stringBuffer.setLength (0);
        do {
            stringBuffer.append ((char)ch);
            ch = fileReader.read();
        } while (Character.isDigit(ch));
        cardData.cardNumber = stringBuffer.toString();

        // Validate and skip spaces

        if (ch != ' ')
            throw new RuntimeException ("invalid char in card data");
        do ch = fileReader.read(); while (ch == ' ');

        // Read the PIN

        if ( ! Character.isDigit(ch))
            throw new RuntimeException ("invalid char in card data");
        stringBuffer.setLength (0);
        do {
            stringBuffer.append ((char)ch);
            ch = fileReader.read();
        } while (Character.isDigit(ch));
        if (stringBuffer.length() != PIN_LEN)
            throw new RuntimeException ("invalid PIN length");
        cardData.cardPIN = stringBuffer.toString();

        // Validate the end of the line

        if (ch == '\r')
            ch = fileReader.read();
        if (ch != '\n')
            throw new RuntimeException ("invalid char in card data");
        ch = fileReader.read();

        // Append the card number-PIN combination to the
        // card data vector

        cardDataVec.add (cardData);
    }

    // Close the card data file

    fileReader.close ();
}

// Process exceptions

catch (IOException e) {
    throw new RuntimeException (e.toString());
}

// Return an array of card data
```

```
    return cardDataVec.toArray(new CardData[1]);
}

//-----------------------------------------------------------------------

// ENTER A STRING INTO THE KEYPAD

static private void
enterString (
    String          str)            // String to be keyed
{
    int             i;              // General purpose index
    char            ch;             // Character to be keyed

    for (i = 0; i < str.length(); i++) {
        if (i != 0) randWait (MIN_INTER_KEY_WAIT, MAX_INTER_KEY_WAIT);
        ch = str.charAt(i);
        switch (ch) {
        case '0':
            ATMInfra.emulateTouch (246, 406);
            break;
        case '.':
            ATMInfra.emulateTouch (394, 406);
            break;
        case '1':
            ATMInfra.emulateTouch (246, 332);
            break;
        case '2':
            ATMInfra.emulateTouch (320, 332);
            break;
        case '3':
            ATMInfra.emulateTouch (394, 332);
            break;
        case '4':
            ATMInfra.emulateTouch (246, 258);
            break;
        case '5':
            ATMInfra.emulateTouch (320, 258);
            break;
        case '6':
            ATMInfra.emulateTouch (394, 258);
            break;
        case '7':
            ATMInfra.emulateTouch (246, 184);
            break;
        case '8':
            ATMInfra.emulateTouch (320, 184);
            break;
        case '9':
            ATMInfra.emulateTouch (394, 184);
            break;
        case '\n':
            ATMInfra.emulateTouch (560, 184);
            break;
        case '\b':
            ATMInfra.emulateTouch (560, 258);
            break;
        default:
            throw new RuntimeException ("invalid touch-screen char");
        }
    }
}
```

```
//-----------------------------------------------------------------------

// COMPARE SCREEN WITH PREVIOUS DUMP

static private boolean
compareScreen (
    long[]          control)        // Control screen dump
{
    long[]          sample;         // Pixel array sample from the screen
    int             mismatchCnt;    // Mismatch count
    int             i;              // General purpose index

    sample = ATMInfra.readScreenPixels();
    mismatchCnt = 0;
    for (i = 0; i < PIX_ARR_SIZE && mismatchCnt <= MAX_CHG_PIX; i++)
        if (sample[i] != control[i]) mismatchCnt ++ ;
    return mismatchCnt <= MAX_CHG_PIX;
}

//-----------------------------------------------------------------------

// MAIN LINE

static public void
main (
    String[]        argv)           // Argument values
{
    int             cardInd;        // Card index
    String          account;        // Account code
    int             amount;         // Amount to dispense
    int             amtPaid;        // Amount paid
    int             totAmtPaid;     // Total amount paid

    // Reset the amount dispensed

    ATMInfra.getAmountPaid ();

    // Loop through withdrawal cycles

    for (;;) {

        // Randomly select a card from the available alternatives

        cardInd = randGen.nextInt (cardDataArr.length);

        // Emulate insertion of the card into the ATM

        ATMInfra.emulateCardIn (cardDataArr[cardInd].cardNumber);

        // Wait for 3 to 7 seconds

        randWait (3000, 7000);

        // Check that the card was accepted

        if ( ! compareScreen (acpCardPix)) {
            System.out.println ("Error: card number " +
                cardDataArr[cardInd].cardNumber +
                " was rejected by the ATM");
            System.exit (1);
        }

        // Enter the PIN
```

```
            enterString (cardDataArr[cardInd].cardPIN);

            // Wait for 1.5 to 3 seconds

            randWait (1500, 3000);

            // Check that the PIN was accepted

            if ( ! compareScreen (acpPINPix)) {
                System.out.println ("Error: PIN for card number " +
                    cardDataArr[cardInd].cardNumber +
                    " was rejected by the ATM");
                System.exit (1);
            }

            // Select the account

            switch (randGen.nextInt (3)) {
            case 0:     ATMInfra.emulateTouch (320, 184);       break;
            case 1:     ATMInfra.emulateTouch (320, 258);       break;
            case 2:     ATMInfra.emulateTouch (320, 332);       break;
            }

            // Wait for 1.5 to 3 seconds

            randWait (1500, 3000);

            // Check that the account selection was accepted

            if ( ! compareScreen (acpAccPix)) {
                System.out.println ("Error: Account for card number " +
                    cardDataArr[cardInd].cardNumber +
                    " was rejected by the ATM");
                System.exit (1);
            }

            // Select the amount to be dispensed, convert
            // it into a string and then key the string

            amount = MIN_AMT +
                randGen.nextInt ((MAX_AMT-MIN_AMT)/DISP_UNIT + 1) * DISP_UNIT;
            enterString (amount + ".00\n");

            // Wait for 5 seconds

            randWait (5000, 5000);

            // Load the amount paid

            totAmtPaid = amtPaid = ATMInfra.getAmountPaid ();
            while (amtPaid != 0) {
                randWait (1000, 1000);
                amtPaid = ATMInfra.getAmountPaid ();
                totAmtPaid += amtPaid;
            }
            if (totAmtPaid != amount) {
                System.out.println ("Error: amount requested=$" + amount
                    + " amount paid=$" + totAmtPaid);
                System.exit (1);
            }
        }
    }
}
```