```
// FailsafeStorage.java - FAILSAFE STORAGE CLASS
//
// MODULE INDEX
// NAME                 CONTENTS
// getCont              Get the contents of a file
// putCont              Update the contents of a file
// read                 Retrieve the data in the failsafe storage
// write                Update the data in the failsafe storage
// FailsafeStorage      Constructor
//
// MAINTENANCE HISTORY
// DATE         PROGRAMMER AND DETAILS
// 03-09-05     JS      Original
//
//-----------------------------------------------------------------------------

// IMPORTS

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

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

class FailsafeStorage
implements FailsafeInterface
{
    //-------------------------------------------------------------------------

    // DEFINITIONS

    private static final String [] FILE_NAME_ARR = {"FAILSAFE_0","FAILSAFE_1"};
                                        // Disk file name array
    private static final int SERIAL_NO_LEN = 4;
                                        // Serial number length in bytes
    private static final int CRC_LEN = 2;
                                        // CRC length in bytes

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

    // FILE CONTENTS STRUCTURE

    class FileCont {
        long            fcSerialNo;     // Serial number
        byte []         fcData;         // File contents data
    }

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

    // CLASS INSTANCE VARIABLES

    private int         fssLastFileNo;  // Last file number to be updated
    private long        fssLastSerialNo; // Last serial number

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

    // GET THE CONTENTS OF A FILE

    private FileCont
    getCont (
        int             fileNo)         // File number (zero or one)
    throws IOException
    {
        FileInputStream fileInputStream; // File input stream reference
        int             fileLength;     // File length
        byte []         fileByteArr;    // File byte array
        FileCont        fileCont;       // File contents instance
        int             dataLength;     // Data length
        int             i;              // General purpose index

        // Open and read the file

        fileInputStream = new FileInputStream (FILE_NAME_ARR[fileNo]);
        fileLength = fileInputStream.available();
        fileByteArr = new byte [ fileLength ];
        if (fileInputStream.read (fileByteArr) != fileLength) {
            fileInputStream.close ();
            throw new IOException ("partial read");
        }
        fileInputStream.close ();
```

```
    // Validate the CRC

    if (Checksums.crc16 (fileByteArr, fileLength) != 0)
        throw new IOException ("CRC error");

    // Load the file contents

    fileCont = new FileCont ();
    fileCont.fcSerialNo = 0;
    for (i = 0; i < SERIAL_NO_LEN; i++)
        fileCont.fcSerialNo = (fileCont.fcSerialNo << 8)
            | (fileByteArr[i] & 0xff);
    dataLength = fileLength - SERIAL_NO_LEN - CRC_LEN;
    fileCont.fcData = new byte [ dataLength ];
    for (i = 0; i < dataLength; i++)
        fileCont.fcData[i] = fileByteArr[i+SERIAL_NO_LEN];

    // Return the file contents

    return fileCont;
}

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

// UPDATE THE CONTENTS OF A FILE

void
putCont (
    int             fileNo,         // File number (zero or one)
    long            serialNo,       // Serial number
    byte []         data,           // Data to write
    int             length)         // Length to write
throws IOException
{
    byte []         fileByteArr;    // File byte array
    int             fileLength;     // File length
    int             i;              // General purpose index
    int             crc;            // 16-bit CRC
    FileOutputStream fileOutputStream; // File output stream reference

    // Load the file contents array

    fileLength = SERIAL_NO_LEN + length + CRC_LEN;
    fileByteArr = new byte [ fileLength ];
    for (i = SERIAL_NO_LEN-1; i >= 0; i--) {
        fileByteArr[i] = (byte)(serialNo & 0xff);
        serialNo >>= 8;
    }
    for (i = 0; i < length; i++)
        fileByteArr[SERIAL_NO_LEN+i] = (byte)(data[i] & 0xff);
    crc = Checksums.crc16 (fileByteArr, SERIAL_NO_LEN+length);
    fileByteArr[fileLength-2] = (byte)(crc & 0xff);
    fileByteArr[fileLength-1] = (byte)((crc >> 8) & 0xff);

    // Write the file contents to the file

    fileOutputStream = new FileOutputStream (FILE_NAME_ARR[fileNo]);
    fileOutputStream.write (fileByteArr);
    fileOutputStream.close ();
}

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

// RETRIEVE THE DATA IN THE FAILSAFE STORAGE

public byte []
read ()
throws IOException
{
    int             i;              // General purpose index
    FileCont        fileCont;       // File contents reference
    FileCont        lastCont;       // Last file contents
    byte []         data;           // Returned data array

    // Attempt to retrieve the latest contents of the disk files

    fssLastFileNo = -1;
    lastCont = null;
```

```
        for (i = 0; i < FILE_NAME_ARR.length; i++) {
            try {
                fileCont = getCont (i);
                if (
                    lastCont == null ||
                    ((fileCont.fcSerialNo - lastCont.fcSerialNo)
                        & 0xffffffffL) < 0x10000000L
                ) {
                    lastCont = fileCont;
                    fssLastFileNo = i;
                }
            }
            catch (IOException e) {
                // Empty
            }
        }

        // If neither file could be read, throw an exception

        if (fssLastFileNo == -1)
            throw new IOException ("neither failsafe file could be read");

        // Update the last serial number

        fssLastSerialNo = lastCont.fcSerialNo;

        // Return the retrieved data

        return lastCont.fcData;
    }

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

    // UPDATE THE DATA IN THE FAILSAFE STORAGE

    public void
    write (
        byte []         data,           // Reference to the data to be written
        int             length)         // Size of the data to be written
    throws IOException
    {
        // Attempt to read the lastest data

        try {
            read ();
        } catch (IOException e) {
            // Empty
        }

        // If neither file could be read, initialise the serial number

        if (fssLastFileNo == -1) fssLastSerialNo = 0;

        // Allocate a new serial number and switch to the alternate file

        fssLastSerialNo = (fssLastSerialNo + 1) & 0xffffffff;
        fssLastFileNo = (fssLastFileNo + 1) % FILE_NAME_ARR.length;

        // Attempt to write the data to the file

        putCont (fssLastFileNo, fssLastSerialNo, data, length);
    }

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

    // CONSTRUCTOR

    FailsafeStorage ()
    {
        fssLastFileNo = -1;
    }
}
```