```
// Schedule.java - SCHEDULING PROGRAM
//
// USAGE
// Schedule input-file
//
// input-file   is the name of the input file.
//
// MODULE INDEX
// NAME                          CONTENTS
// FinishDayComparator.compare  Compare finish day keys
// readToken                    Read a token from the input file
// getDateValue                 Get date value of current input token
// readInput                    Read the input file
// scheduleTask                 Schedule a task
// scheduleProj                 Schedule the project
// createFillers                Create filler tasks
// dayToDate                    Convert project day number into s date string
// title                        Display the report title
// generateReport               Generate the report
// Schedule                     Constructor
// main                         Main line
//
// MAINTENANCE HISTORY
// DATE         PROGRAMMER AND DETAILS
// 02-09-05     JS      Original
// 07-09-05     ELT     Corrected error message for no tasks keyword
// 07-09-05     ELT     Added filler task creation in the beginning of schedule
// 09-10-05     JS      Optimised coding of filler task creation
//
//-----------------------------------------------------------------------------

// IMPORTS

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

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

// SCHEDULER CLASS

public class Schedule
{
    //-------------------------------------------------------------------------

    // DEFINITIONS

    static private final int    PAGE_HEIGHT = 66;
                                        // Page height
    static private final String DATE_FORMAT_PATTERN = "dd-MM-yy";
                                        // Date format pattern
    static private final String DATE_TIME_FORMAT_PATTERN = "dd-MM-yy HH:mm";
                                        // Date format pattern

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

    // TOKEN TYPES

    static private final int    TT_WORD = 0;    // Identifier, date, number
    static private final int    TT_STRING = 1;  // String
    static private final int    TT_SEMI = 2;    // Semicolon
    static private final int    TT_COMMA = 3;   // Comma
    static private final int    TT_OPEN = 4;    // Open braces
    static private final int    TT_CLOSE = 5;   // Close braces
    static private final int    TT_EOF = 6;     // End-of-file

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

    // ERROR CLASS

    class Error
    extends Throwable
    {
        //---------------------------------------------------------------------

        // CLASS INSTANCE VARIABLES

        String          errMsg;         // Error message
```

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

    // CONSTRUCTOR

    Error (
        String              msg)            // Error message
    {
        errMsg = new String(msg);
    }
}

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

// TASK CLASS STRUCTURE

class Task {
    int                     taskNo;         // Task number
    String                  taskName;       // Task name
    String                  taskDescr;      // Task description
    String                  taskPerson;     // Person who is to do the task
    int                     taskDuration;   // Task duration
    Vector                  taskNeeds;      // Task numbers of `needs' list
    int                     taskStartDay;   // Start day number
    int                     taskFinishDay;  // Finish day number
    boolean                 taskScheduled;  // Task scheduled flag
}

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

// FINISH DAY KEY

class FinishDayKey {
    int                     fdkFinishDay;   // Finish day number
    int                     fdkTaskNo;      // Task number
}

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

// COMPARE FINISH DAY KEYS

class FinishDayComparator
implements Comparator
{
    public int
    compare (
        Object              object1,        // First object to compare
        Object              object2)        // Second object to compare
    {
        FinishDayKey        finishDayKey1;  // First finish day key
        FinishDayKey        finishDayKey2;  // Second finish day key

        finishDayKey1 = (FinishDayKey) object1;
        finishDayKey2 = (FinishDayKey) object2;

        if (finishDayKey1.fdkFinishDay < finishDayKey2.fdkFinishDay)
            return -1;
        if (finishDayKey1.fdkFinishDay > finishDayKey2.fdkFinishDay)
            return 1;
        if (finishDayKey1.fdkTaskNo < finishDayKey2.fdkTaskNo)
            return -1;
        if (finishDayKey1.fdkTaskNo > finishDayKey2.fdkTaskNo)
            return 1;
        return 0;
    }
}

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

// CLASS INSTANCE VARIABLES

private String              fileName;       // Input file name
private Date                sysTime;        // System date and time
private LineNumberReader    inReader;       // Input reader with line nos
private int                 inCh;           // Current input character
private int                 inTokenType;    // Current token type
private String              inTokenValue;   // Current token value
private Date                startDate;      // Start date
private TreeSet             holidays;       // List of holidays
private int                 taskCount;      // Number of tasks
```

```
private Vector              taskTable;      // Task table
private TreeMap             taskNameMap;    // Task name map
private TreeMap             lastTaskMap;    // Last task map
private int                 pageNo;         // Page number
private int                 linesRemaining; // Lines remaining on page

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

// READ A TOKEN FROM THE INPUT FILE

private void
readToken ()
throws Exception, Error
{
    // Skip white space and comments

    while (
        inCh != -1 &&
        (Character.isWhitespace ((char)inCh) || inCh == '#')
    ) {
        if (inCh == '#') {
            do
                inCh = inReader.read();
            while (inCh != -1 && inCh != '\n' && inCh != '\r');
        } else {
            inCh = inReader.read();
        }
    }

    // Process single-character tokens

    if (inCh == -1) {
        inTokenType = TT_EOF;
        inCh = inReader.read ();
    }
    else if (inCh == ';') {
        inTokenType = TT_SEMI;
        inCh = inReader.read ();
    }
    else if (inCh == '{') {
        inTokenType = TT_OPEN;
        inCh = inReader.read ();
    }
    else if (inCh == '}') {
        inTokenType = TT_CLOSE;
        inCh = inReader.read ();
    }
    else if (inCh == ',') {
        inTokenType = TT_COMMA;
        inCh = inReader.read ();
    }

    // Process words of various kinds

    else if (Character.isLetterOrDigit((char)inCh) || inCh == '_')  {
        inTokenType = TT_WORD;
        inTokenValue = new String ();
        do {
            inTokenValue += (char)inCh;
            inCh = inReader.read();
        } while (
            inCh != -1 && (
                Character.isLetterOrDigit((char)inCh) ||
                inCh == '_' ||
                inCh == '-'
            )
        );
    }

    // Process strings

    else if (inCh == '"') {
        inTokenType = TT_STRING;
        inCh = inReader.read();
        inTokenValue = new String ();
        while (inCh != -1 && inCh != '"' && inCh != '\n' && inCh != '\r') {
            if (inCh == '\\') {
                inCh = inReader.read();
                if (inCh == -1)
```

```
                    throw new Error ("eof in string");
                if (inCh == '\n' || inCh == '\r')
                    throw new Error ("new line in string");
                if (inCh != '\\' && inCh != '"')
                    inTokenValue += '\\';
            }
            inTokenValue += (char)inCh;
            inCh = inReader.read();
        }
        if (inCh == -1)
            throw new Error ("eof in string");
        if (inCh == '\n' || inCh == '\r')
            throw new Error ("new line in string");
        inCh = inReader.read();
    }

    // Other characters are unacceptable

    else
        throw new Error ("bad input char");
}

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

// GET DATE VALUE OF CURRENT INPUT TOKEN

private Date
getDateValue ()
throws Exception
{
    SimpleDateFormat dateFormat;    // Date formatter

    dateFormat = new SimpleDateFormat (DATE_FORMAT_PATTERN);
    return dateFormat.parse (inTokenValue);
}

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

// READ THE INPUT FILE

private void
readInput ()
throws Error
{
    Task            task;           // Task instance
    Task            lastTask;       // Last task done by this person
    Task            needTask;       // Needed task

    // Open the input file

    try {
        inReader = new LineNumberReader (new FileReader (fileName));
    }
    catch (Exception e) {
        throw new Error (e.toString());
    }

    // Catch exceptions to insert line numbers

    try {
        // Load the look-ahead character

        inCh = inReader.read();

        // Read the look-ahead token

        readToken ();

        // Process the start date

        if (inTokenType != TT_WORD || ! inTokenValue.equals("start"))
            throw new Error ("no start keyword");
        readToken ();
        startDate = getDateValue ();
        readToken ();
        if (inTokenType != TT_SEMI)
            throw new Error ("missing semicolon");
        readToken ();
```

```
// Process the holidays

holidays = new TreeSet ();
if (inTokenType != TT_WORD || ! inTokenValue.equals("holidays"))
    throw new Error ("no holidays keyword");
readToken ();
if (inTokenType != TT_OPEN)
    throw new Error ("no open brace");
readToken ();
if (inTokenType != TT_CLOSE) {
    for (;;) {
        if (inTokenType != TT_WORD)
            throw new Error ("date expected");
        holidays.add (getDateValue());
        readToken ();
        if (inTokenType != TT_COMMA) break;
        readToken ();
    }
    if (inTokenType != TT_CLOSE)
        throw new Error ("no close brace");
}
readToken ();

// Process the task list

taskCount = 0;
taskTable = new Vector ();
taskNameMap = new TreeMap ();
lastTaskMap = new TreeMap ();
if (inTokenType != TT_WORD || ! inTokenValue.equals("tasks"))
    throw new Error ("no tasks keyword");
readToken ();
if (inTokenType != TT_OPEN)
    throw new Error ("no open brace");
readToken ();
while (inTokenType == TT_WORD) {

    // Create a new Task instance

    task = new Task ();

    // Read the task name

    task.taskName = inTokenValue;
    readToken ();

    // Read the task description

    if (inTokenType != TT_STRING)
        throw new Error ("task description expected");
    task.taskDescr = inTokenValue;
    readToken ();

    // Read the initials of the person who will do the task

    if (inTokenType != TT_WORD)
        throw new Error ("person's initials expected");
    task.taskPerson = inTokenValue;
    readToken ();

    // Read the task duration

    if (inTokenType != TT_WORD)
        throw new Error ("person's initials expected");
    task.taskDuration = Integer.parseInt (inTokenValue);
    readToken ();

    // Initialise the needs list

    task.taskNeeds = new Vector ();

    // If the person is known, add the person's last task
    // to the needs list

    lastTask = (Task) lastTaskMap.get (task.taskPerson);
    if (lastTask != null)
        task.taskNeeds.add (lastTask);

    // Add any explicit needs to the needs list
```

```
            if (inTokenType == TT_WORD && inTokenValue.equals("needs")) {
                do {
                    readToken ();
                    if (inTokenType != TT_WORD)
                        throw new Error ("needed task name expected");
                    needTask = (Task) taskNameMap.get (inTokenValue);
                    if (needTask == null)
                        throw new Error ("unrecognised needed task");
                    task.taskNeeds.add (needTask);
                    readToken ();
                } while (inTokenType == TT_COMMA);
            }

            // Reset the task scheduled flag

            task.taskScheduled = false;

            // Add the task to the task table and the task name map

            task.taskNo = taskCount ++ ;
            taskTable.add (task);
            taskNameMap.put (task.taskName, task);
            lastTaskMap.put (task.taskPerson, task);

            // Validate the semicolon at the end of the task definition

            if (inTokenType != TT_SEMI)
                throw new Error ("semicolon expected");
            readToken ();
        }
        if (inTokenType != TT_CLOSE)
            throw new Error ("task name expected");
        readToken ();

        // Check for end-of-file

        if (inTokenType != TT_EOF)
            throw new Error ("data after task list");

        // Close the input file

        inReader.close ();
    }

    catch (Exception e) {
        throw new Error ("line " + (inReader.getLineNumber()+1) + ": "
            + e.toString());
    }

    catch (Error e) {
        throw new Error ("line " + (inReader.getLineNumber()+1) + ": "
            + e.errMsg);
    }
}

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

// SCHEDULE A TASK

private void
scheduleTask (
    Task            task)           // Task to be scheduled
{
    int             startDay;       // Start day number
    Task            needTask;       // Needed task reference
    int             i;              // General purpose index

    // Find the greatest finish day of each task that this task needs

    startDay = 0;
    for (i = 0; i < task.taskNeeds.size(); i++) {
        needTask = (Task) task.taskNeeds.get(i);
        if ( ! needTask.taskScheduled)
            scheduleTask (needTask);
        if (needTask.taskFinishDay >= startDay)
            startDay = needTask.taskFinishDay + 1;
    }
```

```
    // Load the start day and finish days and set the scheduled flag

    task.taskStartDay = startDay;
    task.taskFinishDay = startDay + task.taskDuration - 1;
    task.taskScheduled = true;
}

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

// SCHEDULE THE PROJECT

private void
scheduleProj ()
{
    int             i;              // General purpose index
    Task            task;           // Task reference

    for (i = 0; i < taskTable.size(); i++) {
        task = (Task) taskTable.get(i);
        if ( ! task.taskScheduled) scheduleTask (task);
    }
}

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

// CREATE FILLER TASKS

private void
createFillers ()
{
    Iterator        lastTaskIterator; // Last task map iterator
    String          person;         // Person's name
    TreeMap         startDayMap;    // Start day map
    Iterator        startDayIterator; // Start day map iterator
    int             i;              // General purpose index
    Task            task;           // Task reference
    Task            prevTask;       // Previous task
    Task            fillTask;       // Filler task
    int             prevFinishDay;  // Previous task finish day

    // Process each person from the last task map

    lastTaskIterator = lastTaskMap.keySet().iterator();
    while (lastTaskIterator.hasNext()) {
        person = (String) lastTaskIterator.next();

        // Construct a list of the person's tasks sorted by start day

        startDayMap = new TreeMap ();
        for (i = 0; i < taskTable.size(); i++) {
            task = (Task) taskTable.get(i);
            if (task.taskPerson.equals(person))
                startDayMap.put (new Integer(task.taskStartDay), task);
        }

        // Check for idle time and create filler tasks as needed

        prevTask = null;
        startDayIterator = startDayMap.values().iterator();
        while (startDayIterator.hasNext()) {
            task = (Task) startDayIterator.next();
            prevFinishDay = prevTask == null ? -1 : prevTask.taskFinishDay;
            if ( task.taskStartDay != prevFinishDay + 1) {
                fillTask = new Task ();
                fillTask.taskName = "SPACE";
                fillTask.taskDescr = "Waiting for dependency";
                fillTask.taskPerson = person;
                fillTask.taskDuration = task.taskStartDay-prevFinishDay-1;
                fillTask.taskStartDay = prevFinishDay + 1;
                fillTask.taskFinishDay = task.taskStartDay - 1;
                fillTask.taskNo = taskCount ++ ;
                taskTable.add (fillTask);
            }
            prevTask = task;
        }
    }
}

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

```
// CONVERT PROJECT DAY NUMBER TO A DATE STRING

private String
dayToDate (
    int             day)            // Project day number
{
    Calendar        calendar;       // Calendar corresponding to day number
    SimpleDateFormat dateFormat;    // Date formatter

    // This is not particularly efficient, but it is quite robust.
    // Start at the first day in the project.

    calendar = new GregorianCalendar ();
    calendar.setTime (startDate);
    for (;;) {

        // Skip weekends and public holidays

        while (
            calendar.get(Calendar.DAY_OF_WEEK) == Calendar.SATURDAY ||
            calendar.get(Calendar.DAY_OF_WEEK) == Calendar.SUNDAY ||
            holidays.contains(calendar.getTime())
        ) calendar.add (Calendar.DATE, 1);

        // If there are no more days, exit the loop

        if (day == 0) break;

        // Move to the next day in the project

        calendar.add (Calendar.DATE, 1);
        day -- ;
    }

    // Convert the date in the calendar into a string

    dateFormat = new SimpleDateFormat (DATE_FORMAT_PATTERN);
    return dateFormat.format (calendar.getTime());
}

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

// DISPLAY THE REPORT TITLE

private void
title ()
{
    SimpleDateFormat dateFormat;    // Date formatter
    StringBuffer    buf;            // Formatting buffer
    int             i;              // General purpose index

    // Flush the remainder of the previous page

    if (pageNo != 0) {
        while (linesRemaining > 0) {
            System.out.println ();
            linesRemaining ++ ;
        }
        for (i = 0; i < 6; i++) System.out.println ();
    }

    // Format and display the title line

    dateFormat = new SimpleDateFormat (DATE_TIME_FORMAT_PATTERN);
    buf = new StringBuffer();
    buf.append (dateFormat.format (sysTime));
    while (buf.length() < 27) buf.append (' ');
    buf.append ("PROJECT SCHEDULE LISTING");
    while (buf.length() < 74) buf.append (' ');
    buf.append ("PAGE ");
    buf.append (++pageNo);
    System.out.println (buf);

    // Display `for <file-name>' centred

    buf = new StringBuffer();
    while (buf.length() < (80 - 4 - fileName.length()) / 2) buf.append(' ');
    buf.append ("for ");
```

```
    buf.append (fileName);
    System.out.println (buf);
    System.out.println ();

    // Display the column titles

    buf = new StringBuffer();
    buf.append ("Task name  Description");
    while (buf.length() < 50) buf.append (' ');
    buf.append ("Person Days  Start    Finish");
    System.out.println (buf);
    System.out.println ();

    // Reset the number of lines remaining on the current page

    linesRemaining += PAGE_HEIGHT - 5 - 6;
}

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

// GENERATE THE REPORT

private void
generateReport ()
{
    FinishDayKey    finishDayKey;   // Finish day key reference
    TreeSet         finishDayIndex; // Finish day index
    Iterator        finishDayIterator; // Finish day index iterator
    int             i;              // General purpose index
    Task            task;           // Task reference
    int             projDuration;   // Project duration
    StringBuffer    buf;            // Formatting buffer
    NumberFormat    numberFormat;   // Number formatter

    // Initialise the class instance variables

    sysTime = Calendar.getInstance().getTime();
    pageNo = 0;
    linesRemaining = 0;

    // Load the finish day index to order the tasks by finish day
    // and calculate the project duration

    finishDayIndex = new TreeSet (new FinishDayComparator());
    for (i = 0; i < taskTable.size(); i++) {
        task = (Task) taskTable.get(i);
        finishDayKey = new FinishDayKey ();
        finishDayKey.fdkFinishDay = task.taskFinishDay;
        finishDayKey.fdkTaskNo = i;
        finishDayIndex.add (finishDayKey);
    }

    // Display the first page title

    title ();

    // Display the report lines and determine the project duration

    projDuration = 0;
    finishDayIterator = finishDayIndex.iterator();
    while (finishDayIterator.hasNext()) {
        finishDayKey = (FinishDayKey) finishDayIterator.next();
        task = (Task) taskTable.get(finishDayKey.fdkTaskNo);

        // Start a new page if the last page is full

        if (linesRemaining < 2) title ();

        // Format and display the task line

        buf = new StringBuffer ();
        buf.append (task.taskName);
        while (buf.length() < 11) buf.append (' ');
        buf.append (task.taskDescr);
        while (buf.length() < 52) buf.append (' ');
        buf.append (task.taskPerson);
        while (buf.length() < 58) buf.append (' ');
        buf.append (task.taskDuration);
        while (buf.length() < 61) buf.insert (58, ' ');
```

```
        while (buf.length() < 63) buf.append (' ');
        buf.append (dayToDate(task.taskStartDay));
        while (buf.length() < 72) buf.append (' ');
        buf.append (dayToDate(task.taskFinishDay));
        System.out.println (buf);

        // Decrement the number of lines remaining on the page

        linesRemaining -- ;

        // Update the project duration

        if (task.taskFinishDay + 1 > projDuration)
            projDuration = task.taskFinishDay + 1;
    }

    buf = new StringBuffer ();
    while (buf.length() < 57) buf.append (' ');
    buf.append ("----");
    System.out.println (buf);

    buf = new StringBuffer ();
    while (buf.length() < 57) buf.append (' ');
    buf.append (projDuration);
    while (buf.length() < 61) buf.insert (57, ' ');
    System.out.println (buf);
}

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

// CONSTRUCTOR

Schedule (
    String []       argv)           // Argument values
{
    // Catch errors

    try {

        // Validate the command line and assign the input file name

        if (argv.length != 1) {
            System.out.println ("Usage: Schedule file-name ");
            System.exit (1);
        }
        fileName = argv[0];

        // Read the input

        readInput ();

        // Schedule the project

        scheduleProj ();

        // Create filler pseudo-projects

        createFillers ();

        // Generate the report

        generateReport ();
    }

    // Catch error exceptions

    catch (Error e) {
        System.err.println ("Error: " + e.errMsg);
        System.exit (1);
    }
}

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

// MAIN LINE

static public void
main (
    String []       argv)           // Argument values
```

```
    {
        new Schedule (argv);
    }
}
```