```
Loading vc-cvs...
package caltool.view;

import caltool.caldb.*;
import caltool.schedule.*;
import mvp.*;
import java.util.*;

/****
 *
 * A YearlyCalendar contains a small view for each month, organized in four
 * 3-month rows.  The primarly access interface is through getFirstDay and
 * getNumberOfDays methods.  These methods take a month name and deliver the
 * first day of that month and its number of days, respectively.
 *                                                                          <p>
 * Since the yearly calendar contains no scheduled data itself, there is no
 * need for any model data storage here.  Rather, the access methods consult
 * the calendar db to dynamically generate the date number values for each
 * month.
 *
 */
public class YearlyCalendar extends Model {

    /**
     * Construct this with the given CalendarDB.  Call update to get the data
     * values for the initially current year.
     */
    public YearlyCalendar(CalendarDB calDB) {
        this.calDB = calDB;
        update(null, null);
    }

    /**
     * Return the year number.
     */
    public int getYearNumber() {
        return yearNumber;
    }

    /**
     * Return the first day of the given month.  For initial testing purposes,
     * this method is hard-wired with the sample year shown in Section 2.3.1.4
     * of the requirements.  In the actual implementation, it will consult the
     * calendar db.
     */
    public DayName getFirstDay(MonthName month) {
        switch (month.ordinal()) {
            case 0:
                return DayName.Thursday;
            case 1:
                return DayName.Sunday;
            case 2:
                return DayName.Sunday;
            case 3:
                return DayName.Wednesday;
            case 4:
                return DayName.Friday;
            case 5:
                return DayName.Monday;
            case 6:
                return DayName.Wednesday;
            case 7:
                return DayName.Saturday;
            case 8:
                return DayName.Tuesday;
            case 9:
                return DayName.Thursday;
            case 10:
                return DayName.Sunday;
            case 11:
                return DayName.Tuesday;
        }
        return null; // Cannot happen, but compiler does not know it.
    }

    /**
     * Return the number of weeks in the given month.
     */
    public int getNumberOfWeeks(MonthName month) {
        return (int) Math.ceil(
            ((double)(getNumberOfDays(month) +
                getFirstDay(month).ordinal())) / 7.0);
    }

    /**
     * Return the number of days in the given month.  For initial testing
     * purposes, this method ignores leap years.  In the actual implementation,
     * it will cosult the calendar db to determine the number of days for
     * february in the selected year.
     */
    public int getNumberOfDays(MonthName month) {
        switch (month.ordinal()) {
            case 0:
                return 31;
            case 1:
                return 28;
            case 2:
                return 31;
            case 3:
                return 30;
            case 4:
                return 31;
            case 5:
                return 30;
            case 6:
                return 31;
            case 7:
                return 31;
            case 8:
                return 30;
            case 9:
                return 31;
```

```java
            case 10:
                return 30;
            case 11:
                return 31;
        }
        return -1; // Cannot happen, but compiler does not know it.
    }

    /**
     * Update this' data based on the current selection in the current
     * calendar.  For initial testing purposes, the fixed year of 1998 is
     * created, details of which are shown in Section 2.3.1.4 of the
     * requirements.  In the refined implementation, the calendar db will be
     * consulted to obtain the actual information for the currently selected
     * year.
     */
    public void update(Observable o, Object arg) {

        /*
         * Define fixed data for initial testing purposes.
         */
        yearNumber = 1998;

    }

    /** The number of the year, between 0 and 9999 */
    protected int yearNumber;

    /** The caldb for getting current data */
    CalendarDB calDB;

}
```