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

import mvp.*;
import java.util.Calendar;
import java.text.*;

/****
 *
 * Class Date is the basic unit of calendar time keeping, consisting of a day
 * of the week, numeric date, month, and year.
 */

public class Date extends Model implements Comparable {

    /**
     * Construct an empty Date.
     */
    public Date() {
        day = null;
        number = 0;
        month = null;
        year = 0;
    }

    /**
     * Construct a date from the given string.  Set the valid field to false if
     * the given string does not parse as a valid date.  Note that the invalid
     * state representation is used instead of throwing an exception because
     * some users may want to delay the processing of invalid dates, and hence
     * may not be interested in handling an exception.
     *
     * Use java.text.SimpleDateFormat and java.util.Calendar to do the work.
     * This means that the first time the constructor is invoked, the static
     * format and jCalendar data fields are initialized to new SimpleDateFormat
     * and Calendar objects, resp.  These static values are used in all
     * subsequent Date constructions.
     */
    public Date(String dateString) {

        constructJCalendarIfNecessary();

        try {
            jCalendar.setTime(format.parse(dateString));
            day = convertJavaDay(jCalendar.get(Calendar.DAY_OF_WEEK));
            number = jCalendar.get(Calendar.DAY_OF_MONTH);
            month = MonthName.values()[jCalendar.get(Calendar.MONTH)];
            year = jCalendar.get(Calendar.YEAR);
            jDate = jCalendar.getTime();
            valid = true;
        }
        catch (ParseException e) {
            valid = false;
        }

    }

    /**
     * Construct a date from the given field values.  See the additional
     * comments in the String-valued constructor.
     */
    public Date(DayName day, int number, MonthName month, int year) {

        constructJCalendarIfNecessary();

        this.day = day;
        this.number = number;
        this.month = month;
        this.year = year;

        if (valid =
                (day != null) &&
                (month != null) &&
                (((month == MonthName.January) ||
                  (month == MonthName.March) ||
                  (month == MonthName.May) ||
                  (month == MonthName.July) ||
                  (month == MonthName.August) ||
                  (month == MonthName.October) ||
                  (month == MonthName.December)) ? (number <= 31) :
                      ((month == MonthName.April) ||
                       (month == MonthName.June) ||
                       (month == MonthName.September) ||
                       (month == MonthName.November)) ? (number <= 30) :
                           ((year % 4 == 0) && (year % 400 != 0)) ?
                               (number <= 29) : (number <= 28)) &&
                ((year >= 1) && (year <= 9999))) {
            jCalendar.set(year - 1900, month.ordinal(), number);
            jDate = jCalendar.getTime();
        }
    }

    /**
     * Construct the static java.util.format and Calendar if this is the first
     * time the constructor has been called.
     */
    protected void constructJCalendarIfNecessary() {
        if (format == null) {
            format = (SimpleDateFormat)
                DateFormat.getDateInstance(DateFormat.MEDIUM);
            jCalendar = format.getCalendar();
        }

    }

    /**
     * Convert a java.util.Calendar.DAY_OF_WEEK number to a
     * caltool.schedule.DayName enum.  This is necessary because JFC does not
     * (necessarily) map the pseudo-enum day name literals to any particular
     * numeric sequence.  The last time I checked, Calendar.MONDAY == 2.
     */
    protected DayName convertJavaDay(int javaDayNum) {
```

```java
        switch (javaDayNum) {
            case Calendar.SUNDAY: return DayName.Sunday;
            case Calendar.MONDAY: return DayName.Monday;
            case Calendar.TUESDAY: return DayName.Tuesday;
            case Calendar.WEDNESDAY: return DayName.Wednesday;
            case Calendar.THURSDAY: return DayName.Thursday;
            case Calendar.FRIDAY: return DayName.Friday;
            case Calendar.SATURDAY: return DayName.Saturday;
            default: return null;       // To placate javac
        }
    }

    /**
     * Return true if this is a valid date.
     */
    public boolean isValid() {
        return valid;
    }

    /**
     * Return true if this is an empty date, indicated by the date number = 0.
     */
    public boolean isEmpty() {
        return number == 0;
    }

    /**
     * Return the string representation of this.
     */
    public String toString() {
        return day.toString().concat(" ").concat(Integer.toString(number)).
            concat(" ").concat(month.toString()).concat(" ").
                concat(Integer.toString(year));
    }

    /**
     * Define equality for this as componentwise equality.
     */
    public boolean equals(Object obj) {
        Date otherDate = (Date) obj;

        return
            day.equals(otherDate.day) &&
            number == otherDate.number &&
            month.equals(otherDate.month) &&
            year == otherDate.year;
    }

    /**
     * Define compareTo using java.util.Calendar.  The comparison of invalid
     * dates is defined as follows: (1) invalid < valid; (2) invalid ==
     * invalid.
     *
     */
    public int compareTo(Object o) {
        Date otherDate = (Date) o;

        if ((! valid) && (! otherDate.valid)) {
            return 0;
        }
        if ((! valid) && (otherDate.valid)) {
            return -1;
        }
        if ((valid) && (! otherDate.valid)) {
            return 1;
        }

        /*
         * If both dates are valid, compare using java.util.Date.compareTo.
         */
        return jDate.compareTo(otherDate.jDate);
    }

    /**
     * Define the hash code for this as the sum of the components.  This hash
     * code is used in turn by ItemKey.hashCode.
     */
    public int hashCode() {
        return day.hashCode() + number + month.hashCode() + year;
    }

    /*-*
     * Derived data fields
     */

    /** One of the seven standard days of the week */
    protected DayName day;

    /** Numeric date in a month, between 1 and 31 */
    protected int number;

    /** One of the twelve months of the year */
    protected MonthName month;

    /** The four-digit year number. (Yes, this Calendar Tool has a Y10K
        problem.)
     */
    protected int year;

    /** True if this is a valud date */
    protected boolean valid;

    /** The JFC SimpleDateFormat object to use for date calculations. */
    SimpleDateFormat format;

    /** The JFC  object to use for date calculations. */
    Calendar jCalendar;

    /** The java.util.Date value that represents this' date.  In future, this
        may be the only data rep of this, but for now we keep our own model
        data fields around as well.  At present, the significant use of this
        date rep is in this.compareTo. */
```

```
    protected java.util.Date jDate;

}
```