B

org.jfree.date.SerialDate

B.1. példa
SerialDate.java

```
/* ========================================================================
 * JCommon : a free general purpose class library for the Java(tm) platform
 * ========================================================================
 *
 * (C) Copyright 2000-2005, by Object Refinery Limited and Contributors.
 *
 * Project Info:  http://www.jfree.org/jcommon/index.html
 *
 * This library is free software; you can redistribute it and/or modify it
 * under the terms of the GNU Lesser General Public License as published by
 * the Free Software Foundation; either version 2.1 of the License, or
 * (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
 * or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public
 * License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with this library; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301,
 * USA.
 *
 * [Java is a trademark or registered trademark of Sun Microsystems, Inc.
 * in the United States and other countries.]
 *
 * ---------------
 * SerialDate.java
 * ---------------
 * (C) Copyright 2001-2005, by Object Refinery Limited.
 *
 * Original Author:  David Gilbert (for Object Refinery Limited);
 * Contributor(s):   -;
 *
 * $Id: SerialDate.java,v 1.7 2005/11/03 09:25:17 mungady Exp $
 *
 * Changes (from 11-Oct-2001)
```

B.1. példa
SerialDate.java (folytatás)

```
 * --------------------------
 * 11-Oct-2001 : Re-organised the class and moved it to new package
 *               com.jrefinery.date (DG);
 * 05-Nov-2001 : Added a getDescription() method, and eliminated NotableDate
 *               class (DG);
 * 12-Nov-2001 : IBD requires setDescription() method, now that NotableDate
 *               class is gone (DG);  Changed getPreviousDayOfWeek(),
 *               getFollowingDayOfWeek() and getNearestDayOfWeek() to correct
 *               bugs (DG);
 * 05-Dec-2001 : Fixed bug in SpreadsheetDate class (DG);
 * 29-May-2002 : Moved the month constants into a separate interface
 *               (MonthConstants) (DG);
 * 27-Aug-2002 : Fixed bug in addMonths() method, thanks to N???levka Petr (DG);
 * 03-Oct-2002 : Fixed errors reported by Checkstyle (DG);
 * 13-Mar-2003 : Implemented Serializable (DG);
 * 29-May-2003 : Fixed bug in addMonths method (DG);
 * 04-Sep-2003 : Implemented Comparable.  Updated the isInRange javadocs (DG);
 * 05-Jan-2005 : Fixed bug in addYears() method (1096282) (DG);
 *
 */

package org.jfree.date;

import java.io.Serializable;
import java.text.DateFormatSymbols;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.GregorianCalendar;

/**
 *  An abstract class that defines our requirements for manipulating dates,
 *  without tying down a particular implementation.
 *  <P>
 *  Requirement 1 : match at least what Excel does for dates;
 *  Requirement 2 : class is immutable;
 *  <P>
 *  Why not just use java.util.Date?  We will, when it makes sense.  At times,
 *  java.util.Date can be *too* precise - it represents an instant in time,
 *  accurate to 1/1000th of a second (with the date itself depending on the
 *  time-zone).  Sometimes we just want to represent a particular day (e.g. 21
 *  January 2015) without concerning ourselves about the time of day, or the
 *  time-zone, or anything else.  That's what we've defined SerialDate for.
 *  <P>
 *  You can call getInstance() to get a concrete subclass of SerialDate,
 *  without worrying about the exact implementation.
 *
 * @author David Gilbert
 */
public abstract class SerialDate implements Comparable,
                                            Serializable,
                                            MonthConstants {

    /** For serialization. */
    private static final long serialVersionUID = -293716040467423637L;

    /** Date format symbols. */
    public static final DateFormatSymbols
        DATE_FORMAT_SYMBOLS = new SimpleDateFormat().getDateFormatSymbols();

    /** The serial number for 1 January 1900. */
    public static final int SERIAL_LOWER_BOUND = 2;

    /** The serial number for 31 December 9999. */
    public static final int SERIAL_UPPER_BOUND = 2958465;

```

B.1. példa
SerialDate.java (folytatás)

```
    /** The lowest year value supported by this date format. */
    public static final int MINIMUM_YEAR_SUPPORTED = 1900;

    /** The highest year value supported by this date format. */
    public static final int MAXIMUM_YEAR_SUPPORTED = 9999;

    /** Useful constant for Monday. Equivalent to java.util.Calendar.MONDAY. */
    public static final int MONDAY = Calendar.MONDAY;

    /**
     * Useful constant for Tuesday. Equivalent to java.util.Calendar.TUESDAY.
     */
    public static final int TUESDAY = Calendar.TUESDAY;

    /**
     * Useful constant for Wednesday. Equivalent to
     * java.util.Calendar.WEDNESDAY.
     */
    public static final int WEDNESDAY = Calendar.WEDNESDAY;

    /**
     * Useful constant for Thrusday. Equivalent to java.util.Calendar.THURSDAY.
     */
    public static final int THURSDAY = Calendar.THURSDAY;

    /** Useful constant for Friday. Equivalent to java.util.Calendar.FRIDAY. */
    public static final int FRIDAY = Calendar.FRIDAY;

    /**
     * Useful constant for Saturday. Equivalent to java.util.Calendar.SATURDAY.
     */
    public static final int SATURDAY = Calendar.SATURDAY;

    /** Useful constant for Sunday. Equivalent to java.util.Calendar.SUNDAY. */
    public static final int SUNDAY = Calendar.SUNDAY;

    /** The number of days in each month in non leap years. */
    static final int[] LAST_DAY_OF_MONTH =
        {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

    /** The number of days in a (non-leap) year up to the end of each month. */
    static final int[] AGGREGATE_DAYS_TO_END_OF_MONTH =
        {0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334, 365};

    /** The number of days in a year up to the end of the preceding month. */
    static final int[] AGGREGATE_DAYS_TO_END_OF_PRECEDING_MONTH =
        {0, 0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334, 365};

    /** The number of days in a leap year up to the end of each month. */
    static final int[] LEAP_YEAR_AGGREGATE_DAYS_TO_END_OF_MONTH =
        {0, 31, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335, 366};

    /**
     * The number of days in a leap year up to the end of the preceding month.
     */
    static final int[]
        LEAP_YEAR_AGGREGATE_DAYS_TO_END_OF_PRECEDING_MONTH =
            {0, 0, 31, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335, 366};

    /** A useful constant for referring to the first week in a month. */
    public static final int FIRST_WEEK_IN_MONTH = 1;

```

B.1. példa
SerialDate.java (folytatás)

```
    /** A useful constant for referring to the second week in a month. */
    public static final int SECOND_WEEK_IN_MONTH = 2;

    /** A useful constant for referring to the third week in a month. */
    public static final int THIRD_WEEK_IN_MONTH = 3;

    /** A useful constant for referring to the fourth week in a month. */
    public static final int FOURTH_WEEK_IN_MONTH = 4;

    /** A useful constant for referring to the last week in a month. */
    public static final int LAST_WEEK_IN_MONTH = 0;

    /** Useful range constant. */
    public static final int INCLUDE_NONE = 0;

    /** Useful range constant. */
    public static final int INCLUDE_FIRST = 1;

    /** Useful range constant. */
    public static final int INCLUDE_SECOND = 2;

    /** Useful range constant. */
    public static final int INCLUDE_BOTH = 3;

    /**
     * Useful constant for specifying a day of the week relative to a fixed
     * date.
     */
    public static final int PRECEDING = -1;

    /**
     * Useful constant for specifying a day of the week relative to a fixed
     * date.
     */
    public static final int NEAREST = 0;

    /**
     * Useful constant for specifying a day of the week relative to a fixed
     * date.
     */
    public static final int FOLLOWING = 1;

    /** A description for the date. */
    private String description;

    /**
     * Default constructor.
     */
    protected SerialDate() {
    }

    /**
     * Returns <code>true</code> if the supplied integer code represents a
     * valid day-of-the-week, and <code>false</code> otherwise.
     *
     * @param code  the code being checked for validity.
     *
     * @return <code>true</code> if the supplied integer code represents a
     *         valid day-of-the-week, and <code>false</code> otherwise.
     */
    public static boolean isValidWeekdayCode(final int code) {

```

B.1. példa
SerialDate.java (folytatás)

```
        switch(code) {
            case SUNDAY:
            case MONDAY:
            case TUESDAY:
            case WEDNESDAY:
            case THURSDAY:
            case FRIDAY:
            case SATURDAY:
                return true;
            default:
                return false;
        }

    }

    /**
     * Converts the supplied string to a day of the week.
     *
     * @param s  a string representing the day of the week.
     *
     * @return <code>-1</code> if the string is not convertable, the day of
     *         the week otherwise.
     */
    public static int stringToWeekdayCode(String s) {

        final String[] shortWeekdayNames
            = DATE_FORMAT_SYMBOLS.getShortWeekdays();
        final String[] weekDayNames = DATE_FORMAT_SYMBOLS.getWeekdays();

        int result = -1;
        s = s.trim();
        for (int i = 0; i < weekDayNames.length; i++) {
            if (s.equals(shortWeekdayNames[i])) {
                result = i;
                break;
            }
            if (s.equals(weekDayNames[i])) {
                result = i;
                break;
            }
        }
        return result;

    }

    /**
     * Returns a string representing the supplied day-of-the-week.
     * <P>
     * Need to find a better approach.
     *
     * @param weekday  the day of the week.
     *
     * @return a string representing the supplied day-of-the-week.
     */
    public static String weekdayCodeToString(final int weekday) {

        final String[] weekdays = DATE_FORMAT_SYMBOLS.getWeekdays();
        return weekdays[weekday];

    }

    /**
```

B.1. példa
SerialDate.java (folytatás)

```
     * Returns an array of month names.
     *
     * @return an array of month names.
     */
    public static String[] getMonths() {

        return getMonths(false);
    }

    /**
     * Returns an array of month names.
     *
     * @param shortened  a flag indicating that shortened month names should
     *                   be returned.
     *
     * @return an array of month names.
     */
    public static String[] getMonths(final boolean shortened) {

        if (shortened) {
            return DATE_FORMAT_SYMBOLS.getShortMonths();
        }
        else {
            return DATE_FORMAT_SYMBOLS.getMonths();
        }

    }

    /**
     * Returns true if the supplied integer code represents a valid month.
     *
     * @param code  the code being checked for validity.
     *
     * @return <code>true</code> if the supplied integer code represents a
     *         valid month.
     */
    public static boolean isValidMonthCode(final int code) {

        switch(code) {
            case JANUARY:
            case FEBRUARY:
            case MARCH:
            case APRIL:
            case MAY:
            case JUNE:
            case JULY:
            case AUGUST:
            case SEPTEMBER:
            case OCTOBER:
            case NOVEMBER:
            case DECEMBER:
                return true;
            default:
                return false;
        }

    }

    /**
     * Returns the quarter for the specified month.
     *
```

B.1. példa
SerialDate.java (folytatás)

```
     * @param code  the month code (1-12).
     *
     * @return the quarter that the month belongs to.
     * @throws java.lang.IllegalArgumentException
     */
    public static int monthCodeToQuarter(final int code) {

        switch(code) {
            case JANUARY:
            case FEBRUARY:
            case MARCH: return 1;
            case APRIL:
            case MAY:
            case JUNE: return 2;
            case JULY:
            case AUGUST:
            case SEPTEMBER: return 3;
            case OCTOBER:
            case NOVEMBER:
            case DECEMBER: return 4;
            default: throw new IllegalArgumentException(
                "SerialDate.monthCodeToQuarter: invalid month code.");
        }

    }

    /**
     * Returns a string representing the supplied month.
     * <P>
     * The string returned is the long form of the month name taken from the
     * default locale.
     *
     * @param month  the month.
     *
     * @return a string representing the supplied month.
     */
    public static String monthCodeToString(final int month) {

        return monthCodeToString(month, false);

    }

    /**
     * Returns a string representing the supplied month.
     * <P>
     * The string returned is the long or short form of the month name taken
     * from the default locale.
     *
     * @param month  the month.
     * @param shortened  if <code>true</code> return the abbreviation of the
     *                   month.
     *
     * @return a string representing the supplied month.
     * @throws java.lang.IllegalArgumentException
     */
    public static String monthCodeToString(final int month,
                                           final boolean shortened) {

        // check arguments...
        if (!isValidMonthCode(month)) {
            throw new IllegalArgumentException(
                "SerialDate.monthCodeToString: month outside valid range.");
```

B.1. példa
SerialDate.java (folytatás)

```
        }

        final String[] months;

        if (shortened) {
            months = DATE_FORMAT_SYMBOLS.getShortMonths();
        }
        else {
            months = DATE_FORMAT_SYMBOLS.getMonths();
        }

        return months[month - 1];

    }

    /**
     * Converts a string to a month code.
     * <P>
     * This method will return one of the constants JANUARY, FEBRUARY, ...,
     * DECEMBER that corresponds to the string.  If the string is not
     * recognised, this method returns -1.
     *
     * @param s  the string to parse.
     *
     * @return <code>-1</code> if the string is not parseable, the month of the
     *         year otherwise.
     */
    public static int stringToMonthCode(String s) {

        final String[] shortMonthNames = DATE_FORMAT_SYMBOLS.getShortMonths();
        final String[] monthNames = DATE_FORMAT_SYMBOLS.getMonths();

        int result = -1;
        s = s.trim();

        // first try parsing the string as an integer (1-12)...
        try {
            result = Integer.parseInt(s);
        }
        catch (NumberFormatException e) {
            // suppress
        }

        // now search through the month names...
        if ((result < 1) || (result > 12)) {
            for (int i = 0; i < monthNames.length; i++) {
                if (s.equals(shortMonthNames[i])) {
                    result = i + 1;
                    break;
                }
                if (s.equals(monthNames[i])) {
                    result = i + 1;
                    break;
                }
            }
        }

        return result;

    }

    /**
```

B.1. példa
SerialDate.java (folytatás)

```
     * Returns true if the supplied integer code represents a valid
     * week-in-the-month, and false otherwise.
     *
     * @param code  the code being checked for validity.
     * @return <code>true</code> if the supplied integer code represents a
     *         valid week-in-the-month.
     */
    public static boolean isValidWeekInMonthCode(final int code) {

        switch(code) {
            case FIRST_WEEK_IN_MONTH:
            case SECOND_WEEK_IN_MONTH:
            case THIRD_WEEK_IN_MONTH:
            case FOURTH_WEEK_IN_MONTH:
            case LAST_WEEK_IN_MONTH: return true;
            default: return false;
        }

    }

    /**
     * Determines whether or not the specified year is a leap year.
     *
     * @param yyyy  the year (in the range 1900 to 9999).
     *
     * @return <code>true</code> if the specified year is a leap year.
     */
    public static boolean isLeapYear(final int yyyy) {

        if ((yyyy % 4) != 0) {
            return false;
        }
        else if ((yyyy % 400) == 0) {
            return true;
        }
        else if ((yyyy % 100) == 0) {
            return false;
        }
        else {
            return true;
        }

    }

    /**
     * Returns the number of leap years from 1900 to the specified year
     * INCLUSIVE.
     * <P>
     * Note that 1900 is not a leap year.
     *
     * @param yyyy  the year (in the range 1900 to 9999).
     *
     * @return the number of leap years from 1900 to the specified year.
     */
    public static int leapYearCount(final int yyyy) {

        final int leap4 = (yyyy - 1896) / 4;
        final int leap100 = (yyyy - 1800) / 100;
        final int leap400 = (yyyy - 1600) / 400;
        return leap4 - leap100 + leap400;

    }
```

B.1. példa
SerialDate.java (folytatás)

```
    /**
     * Returns the number of the last day of the month, taking into account
     * leap years.
     *
     * @param month  the month.
     * @param yyyy  the year (in the range 1900 to 9999).
     *
     * @return the number of the last day of the month.
     */
    public static int lastDayOfMonth(final int month, final int yyyy) {

        final int result = LAST_DAY_OF_MONTH[month];
        if (month != FEBRUARY) {
            return result;
        }
        else if (isLeapYear(yyyy)) {
            return result + 1;
        }
        else {
            return result;
        }

    }

    /**
     * Creates a new date by adding the specified number of days to the base
     * date.
     *
     * @param days  the number of days to add (can be negative).
     * @param base  the base date.
     *
     * @return a new date.
     */
    public static SerialDate addDays(final int days, final SerialDate base) {

        final int serialDayNumber = base.toSerial() + days;
        return SerialDate.createInstance(serialDayNumber);

    }

    /**
     * Creates a new date by adding the specified number of months to the base
     * date.
     * <P>
     * If the base date is close to the end of the month, the day on the result
     * may be adjusted slightly:  31 May + 1 month = 30 June.
     *
     * @param months  the number of months to add (can be negative).
     * @param base  the base date.
     *
     * @return a new date.
     */
    public static SerialDate addMonths(final int months,
                                       final SerialDate base) {

        final int yy = (12 * base.getYYYY() + base.getMonth() + months - 1)
                       / 12;
        final int mm = (12 * base.getYYYY() + base.getMonth() + months - 1)
                       % 12 + 1;
        final int dd = Math.min(
            base.getDayOfMonth(), SerialDate.lastDayOfMonth(mm, yy)
```

B.1. példa
SerialDate.java (folytatás)

```
        );
        return SerialDate.createInstance(dd, mm, yy);

    }

    /**
     * Creates a new date by adding the specified number of years to the base
     * date.
     *
     * @param years  the number of years to add (can be negative).
     * @param base  the base date.
     *
     * @return A new date.
     */
    public static SerialDate addYears(final int years, final SerialDate base) {

        final int baseY = base.getYYYY();
        final int baseM = base.getMonth();
        final int baseD = base.getDayOfMonth();

        final int targetY = baseY + years;
        final int targetD = Math.min(
            baseD, SerialDate.lastDayOfMonth(baseM, targetY)
        );

        return SerialDate.createInstance(targetD, baseM, targetY);

    }

    /**
     * Returns the latest date that falls on the specified day-of-the-week and
     * is BEFORE the base date.
     *
     * @param targetWeekday  a code for the target day-of-the-week.
     * @param base  the base date.
     *
     * @return the latest date that falls on the specified day-of-the-week and
     *         is BEFORE the base date.
     */
    public static SerialDate getPreviousDayOfWeek(final int targetWeekday,
                                                  final SerialDate base) {

        // check arguments...
        if (!SerialDate.isValidWeekdayCode(targetWeekday)) {
            throw new IllegalArgumentException(
                "Invalid day-of-the-week code."
            );
        }

        // find the date...
        final int adjust;
        final int baseDOW = base.getDayOfWeek();
        if (baseDOW > targetWeekday) {
            adjust = Math.min(0, targetWeekday - baseDOW);
        }
        else {
            adjust = -7 + Math.max(0, targetWeekday - baseDOW);
        }

        return SerialDate.addDays(adjust, base);

    }
```