



## org.jfree.date.SerialDate

### B.1. példa SerialDate.java

```
1 /* =====
2  * JCommon : a free general purpose class library for the Java(tm) platform
3  * =====
4  *
5  * (C) Copyright 2000-2005, by Object Refinery Limited and Contributors.
6  *
7  * Project Info: http://www.jfree.org/jcommon/index.html
8  *
9  * This library is free software; you can redistribute it and/or modify it
10 * under the terms of the GNU Lesser General Public License as published by
11 * the Free Software Foundation; either version 2.1 of the License, or
12 * (at your option) any later version.
13 *
14 * This library is distributed in the hope that it will be useful, but
15 * WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
16 * or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public
17 * License for more details.
18 *
19 * You should have received a copy of the GNU Lesser General Public
20 * License along with this library; if not, write to the Free Software
21 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301,
22 * USA.
23 *
24 * [Java is a trademark or registered trademark of Sun Microsystems, Inc.
25 * in the United States and other countries.]
26 *
27 * -----
28 * SerialDate.java
29 * -----
30 * (C) Copyright 2001-2005, by Object Refinery Limited.
31 *
32 * Original Author: David Gilbert (for Object Refinery Limited);
33 * Contributor(s): -;
34 *
35 * $Id: SerialDate.java,v 1.7 2005/11/03 09:25:17 mungady Exp $
36 *
37 * Changes (from 11-Oct-2001)
```

## B.1. példa

## SerialDate.java (folytatás)

```

38 * -----
39 * 11-Oct-2001 : Re-organised the class and moved it to new package
40 *               com.jrefinery.date (DG);
41 * 05-Nov-2001 : Added a getDescription() method, and eliminated NotableDate
42 *               class (DG);
43 * 12-Nov-2001 : IBD requires setDescription() method, now that NotableDate
44 *               class is gone (DG); Changed getPreviousDayOfWeek(),
45 *               getFollowingDayOfWeek() and getNearestDayOfWeek() to correct
46 *               bugs (DG);
47 * 05-Dec-2001 : Fixed bug in SpreadsheetDate class (DG);
48 * 29-May-2002 : Moved the month constants into a separate interface
49 *               (MonthConstants) (DG);
50 * 27-Aug-2002 : Fixed bug in addMonths() method, thanks to N???levka Petr (DG);
51 * 03-Oct-2002 : Fixed errors reported by Checkstyle (DG);
52 * 13-Mar-2003 : Implemented Serializable (DG);
53 * 29-May-2003 : Fixed bug in addMonths method (DG);
54 * 04-Sep-2003 : Implemented Comparable. Updated the isInRange javadocs (DG);
55 * 05-Jan-2005 : Fixed bug in addYears() method (1096282) (DG);
56 *
57 */
58
59 package org.jfree.date;
60
61 import java.io.Serializable;
62 import java.text.DateFormatSymbols;
63 import java.text.SimpleDateFormat;
64 import java.util.Calendar;
65 import java.util.GregorianCalendar;
66
67 /**
68 * An abstract class that defines our requirements for manipulating dates,
69 * without tying down a particular implementation.
70 * <P>
71 * Requirement 1 : match at least what Excel does for dates;
72 * Requirement 2 : class is immutable;
73 * <P>
74 * Why not just use java.util.Date? We will, when it makes sense. At times,
75 * java.util.Date can be *too* precise - it represents an instant in time,
76 * accurate to 1/1000th of a second (with the date itself depending on the
77 * time-zone). Sometimes we just want to represent a particular day (e.g. 21
78 * January 2015) without concerning ourselves about the time of day, or the
79 * time-zone, or anything else. That's what we've defined SerialDate for.
80 * <P>
81 * You can call getInstance() to get a concrete subclass of SerialDate,
82 * without worrying about the exact implementation.
83 *
84 * @author David Gilbert
85 */
86 public abstract class SerialDate implements Comparable,
87                                             Serializable,
88                                             MonthConstants {
89
90     /** For serialization. */
91     private static final long serialVersionUID = -293716040467423637L;
92
93     /** Date format symbols. */
94     public static final DateFormatSymbols
95         DATE_FORMAT_SYMBOLS = new SimpleDateFormat().getDateFormatSymbols();
96
97     /** The serial number for 1 January 1900. */
98     public static final int SERIAL_LOWER_BOUND = 2;
99
100    /** The serial number for 31 December 9999. */
101    public static final int SERIAL_UPPER_BOUND = 2958465;
102

```



### B.1. példa

#### SerialDate.java (folytatás)

```

103  /** The lowest year value supported by this date format. */
104  public static final int MINIMUM_YEAR_SUPPORTED = 1900;
105
106  /** The highest year value supported by this date format. */
107  public static final int MAXIMUM_YEAR_SUPPORTED = 9999;
108
109  /** Useful constant for Monday. Equivalent to java.util.Calendar.MONDAY. */
110  public static final int MONDAY = Calendar.MONDAY;
111
112  /**
113   * Useful constant for Tuesday. Equivalent to java.util.Calendar.TUESDAY.
114   */
115  public static final int TUESDAY = Calendar.TUESDAY;
116
117  /**
118   * Useful constant for Wednesday. Equivalent to
119   * java.util.Calendar.WEDNESDAY.
120   */
121  public static final int WEDNESDAY = Calendar.WEDNESDAY;
122
123  /**
124   * Useful constant for Thursday. Equivalent to java.util.Calendar.THURSDAY.
125   */
126  public static final int THURSDAY = Calendar.THURSDAY;
127
128  /** Useful constant for Friday. Equivalent to java.util.Calendar.FRIDAY. */
129  public static final int FRIDAY = Calendar.FRIDAY;
130
131  /**
132   * Useful constant for Saturday. Equivalent to java.util.Calendar.SATURDAY.
133   */
134  public static final int SATURDAY = Calendar.SATURDAY;
135
136  /** Useful constant for Sunday. Equivalent to java.util.Calendar.SUNDAY. */
137  public static final int SUNDAY = Calendar.SUNDAY;
138
139  /** The number of days in each month in non leap years. */
140  static final int[] LAST_DAY_OF_MONTH =
141      {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
142
143  /** The number of days in a (non-leap) year up to the end of each month. */
144  static final int[] AGGREGATE_DAYS_TO_END_OF_MONTH =
145      {0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334, 365};
146
147  /** The number of days in a year up to the end of the preceding month. */
148  static final int[] AGGREGATE_DAYS_TO_END_OF_PRECEDING_MONTH =
149      {0, 0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334, 365};
150
151  /** The number of days in a leap year up to the end of each month. */
152  static final int[] LEAP_YEAR_AGGREGATE_DAYS_TO_END_OF_MONTH =
153      {0, 31, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335, 366};
154
155  /**
156   * The number of days in a leap year up to the end of the preceding month.
157   */
158  static final int[]
159      LEAP_YEAR_AGGREGATE_DAYS_TO_END_OF_PRECEDING_MONTH =
160      {0, 0, 31, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335, 366};
161
162  /** A useful constant for referring to the first week in a month. */
163  public static final int FIRST_WEEK_IN_MONTH = 1;
164

```

## B.1. példa

## SerialDate.java (folytatás)

```

165  /** A useful constant for referring to the second week in a month. */
166  public static final int SECOND_WEEK_IN_MONTH = 2;
167
168  /** A useful constant for referring to the third week in a month. */
169  public static final int THIRD_WEEK_IN_MONTH = 3;
170
171  /** A useful constant for referring to the fourth week in a month. */
172  public static final int FOURTH_WEEK_IN_MONTH = 4;
173
174  /** A useful constant for referring to the last week in a month. */
175  public static final int LAST_WEEK_IN_MONTH = 0;
176
177  /** Useful range constant. */
178  public static final int INCLUDE_NONE = 0;
179
180  /** Useful range constant. */
181  public static final int INCLUDE_FIRST = 1;
182
183  /** Useful range constant. */
184  public static final int INCLUDE_SECOND = 2;
185
186  /** Useful range constant. */
187  public static final int INCLUDE_BOTH = 3;
188
189  /**
190   * Useful constant for specifying a day of the week relative to a fixed
191   * date.
192   */
193  public static final int PRECEDING = -1;
194
195  /**
196   * Useful constant for specifying a day of the week relative to a fixed
197   * date.
198   */
199  public static final int NEAREST = 0;
200
201  /**
202   * Useful constant for specifying a day of the week relative to a fixed
203   * date.
204   */
205  public static final int FOLLOWING = 1;
206
207  /** A description for the date. */
208  private String description;
209
210  /**
211   * Default constructor.
212   */
213  protected SerialDate() {
214  }
215
216  /**
217   * Returns true if the supplied integer code represents a
218   * valid day-of-the-week, and false otherwise.
219   *
220   * @param code the code being checked for validity.
221   *
222   * @return true if the supplied integer code represents a
223   *         valid day-of-the-week, and false otherwise.
224   */
225  public static boolean isValidWeekdayCode(final int code) {
226

```



### B.1. példa SerialDate.java (folytatás)

```

227     switch(code) {
228         case SUNDAY:
229         case MONDAY:
230         case TUESDAY:
231         case WEDNESDAY:
232         case THURSDAY:
233         case FRIDAY:
234         case SATURDAY:
235             return true;
236         default:
237             return false;
238     }
239
240 )
241
242 /**
243  * Converts the supplied string to a day of the week.
244  *
245  * @param s a string representing the day of the week.
246  *
247  * @return <code>-1</code> if the string is not convertible, the day of
248  *         the week otherwise.
249  */
250 public static int stringToWeekdayCode(String s) {
251
252     final String[] shortWeekdayNames
253         = DATE_FORMAT_SYMBOLS.getShortWeekdays();
254     final String[] weekdayNames = DATE_FORMAT_SYMBOLS.getWeekdays();
255
256     int result = -1;
257     s = s.trim();
258     for (int i = 0; i < weekdayNames.length; i++) {
259         if (s.equals(shortWeekdayNames[i])) {
260             result = i;
261             break;
262         }
263         if (s.equals(weekdayNames[i])) {
264             result = i;
265             break;
266         }
267     }
268     return result;
269 }
270
271
272 /**
273  * Returns a string representing the supplied day-of-the-week.
274  * <P>
275  * Need to find a better approach.
276  *
277  * @param weekday the day of the week.
278  *
279  * @return a string representing the supplied day-of-the-week.
280  */
281 public static String weekdayCodeToString(final int weekday) {
282
283     final String[] weekdays = DATE_FORMAT_SYMBOLS.getWeekdays();
284     return weekdays[weekday];
285 }
286
287
288 /**

```

## B.1. példa

## SerialDate.java (folytatás)

```

289  * Returns an array of month names.
290  *
291  * @return an array of month names.
292  */
293  public static String[] getMonths() {
294
295      return getMonths(false);
296  }
297
298  /**
299  * Returns an array of month names.
300  *
301  * @param shortened a flag indicating that shortened month names should
302  *                 be returned.
303  *
304  * @return an array of month names.
305  */
306  public static String[] getMonths(final boolean shortened) {
307
308      if (shortened) {
309          return DATE_FORMAT_SYMBOLS.getShortMonths();
310      }
311      else {
312          return DATE_FORMAT_SYMBOLS.getMonths();
313      }
314  }
315
316  /**
317  * Returns true if the supplied integer code represents a valid month.
318  *
319  * @param code the code being checked for validity.
320  *
321  * @return <code>true</code> if the supplied integer code represents a
322  *         valid month.
323  */
324  public static boolean isValidMonthCode(final int code) {
325
326      switch (code) {
327          case JANUARY:
328          case FEBRUARY:
329          case MARCH:
330          case APRIL:
331          case MAY:
332          case JUNE:
333          case JULY:
334          case AUGUST:
335          case SEPTEMBER:
336          case OCTOBER:
337          case NOVEMBER:
338          case DECEMBER:
339              return true;
340          default:
341              return false;
342      }
343  }
344
345  /**
346  * Returns the quarter for the specified month.
347  *
348  *
349  *
350  */

```



## B.1. példa

## SerialDate.java (folytatás)

```

351  * @param code the month code (1-12).
352  *
353  * @return the quarter that the month belongs to.
354  * @throws java.lang.IllegalArgumentException
355  */
356  public static int monthCodeToQuarter(final int code) {
357
358      switch(code) {
359          case JANUARY:
360          case FEBRUARY:
361          case MARCH: return 1;
362          case APRIL:
363          case MAY:
364          case JUNE: return 2;
365          case JULY:
366          case AUGUST:
367          case SEPTEMBER: return 3;
368          case OCTOBER:
369          case NOVEMBER:
370          case DECEMBER: return 4;
371          default: throw new IllegalArgumentException(
372              "SerialDate.monthCodeToQuarter: invalid month code.");
373      }
374
375  }
376
377  /**
378   * Returns a string representing the supplied month.
379   * <P>
380   * The string returned is the long form of the month name taken from the
381   * default locale.
382   *
383   * @param month the month.
384   *
385   * @return a string representing the supplied month.
386   */
387  public static String monthCodeToString(final int month) {
388
389      return monthCodeToString(month, false);
390
391  }
392
393  /**
394   * Returns a string representing the supplied month.
395   * <P>
396   * The string returned is the long or short form of the month name taken
397   * from the default locale.
398   *
399   * @param month the month.
400   * @param shortened if <code>>true</code> return the abbreviation of the
401   * month.
402   *
403   * @return a string representing the supplied month.
404   * @throws java.lang.IllegalArgumentException
405   */
406  public static String monthCodeToString(final int month,
407                                         final boolean shortened) {
408
409      // check arguments...
410      if (!isValidMonthCode(month)) {
411          throw new IllegalArgumentException(
412              "SerialDate.monthCodeToString: month outside valid range.");

```

# B.1. példa

## SerialDate.java (folytatás)

```

413     )
414
415     final String[] months;
416
417     if (shortened) {
418         months = DATE_FORMAT_SYMBOLS.getShortMonths();
419     }
420     else {
421         months = DATE_FORMAT_SYMBOLS.getMonths();
422     }
423
424     return months[month - 1];
425
426 }
427
428 /**
429  * Converts a string to a month code.
430  * <P>
431  * This method will return one of the constants JANUARY, FEBRUARY, ...,
432  * DECEMBER that corresponds to the string. If the string is not
433  * recognised, this method returns -1.
434  *
435  * @param s the string to parse.
436  *
437  * @return <code>-1</code> if the string is not parseable, the month of the
438  *         year otherwise.
439  */
440 public static int stringToMonthCode(String s) {
441
442     final String[] shortMonthNames = DATE_FORMAT_SYMBOLS.getShortMonths();
443     final String[] monthNames = DATE_FORMAT_SYMBOLS.getMonths();
444
445     int result = -1;
446     s = s.trim();
447
448     // first try parsing the string as an integer (1-12)...
449     try {
450         result = Integer.parseInt(s);
451     }
452     catch (NumberFormatException e) {
453         // suppress
454     }
455
456     // now search through the month names...
457     if ((result < 1) || (result > 12)) {
458         for (int i = 0; i < monthNames.length; i++) {
459             if (s.equals(shortMonthNames[i])) {
460                 result = i + 1;
461                 break;
462             }
463             if (s.equals(monthNames[i])) {
464                 result = i + 1;
465                 break;
466             }
467         }
468     }
469
470     return result;
471
472 }
473
474 /**

```



### B.1. példa

#### SerialDate.java (folytatás)

```

475  * Returns true if the supplied integer code represents a valid
476  * week-in-the-month, and false otherwise.
477  *
478  * @param code the code being checked for validity.
479  * @return <code>true</code> if the supplied integer code represents a
480  *         valid week-in-the-month.
481  */
482  public static boolean isValidWeekInMonthCode(final int code) {
483
484      switch(code) {
485          case FIRST_WEEK_IN_MONTH:
486          case SECOND_WEEK_IN_MONTH:
487          case THIRD_WEEK_IN_MONTH:
488          case FOURTH_WEEK_IN_MONTH:
489          case LAST_WEEK_IN_MONTH: return true;
490          default: return false;
491      }
492  }
493
494  /**
495   * Determines whether or not the specified year is a leap year.
496   *
497   * @param yyyy the year (in the range 1900 to 9999).
498   *
499   * @return <code>true</code> if the specified year is a leap year.
500   */
501  public static boolean isLeapYear(final int yyyy) {
502
503      if ((yyyy % 4) != 0) {
504          return false;
505      }
506      else if ((yyyy % 400) == 0) {
507          return true;
508      }
509      else if ((yyyy % 100) == 0) {
510          return false;
511      }
512      else {
513          return true;
514      }
515  }
516
517  /**
518   * Returns the number of leap years from 1900 to the specified year
519   * INCLUSIVE.
520   * <P>
521   * Note that 1900 is not a leap year.
522   *
523   * @param yyyy the year (in the range 1900 to 9999).
524   *
525   * @return the number of leap years from 1900 to the specified year.
526   */
527  public static int leapYearCount(final int yyyy) {
528
529      final int leap4 = (yyyy - 1896) / 4;
530      final int leap100 = (yyyy - 1800) / 100;
531      final int leap400 = (yyyy - 1600) / 400;
532      return leap4 - leap100 + leap400;
533  }
534
535  }

```

### B.1. példa

#### SerialDate.java (folytatás)

```

537
538 /**
539  * Returns the number of the last day of the month, taking into account
540  * leap years.
541  *
542  * @param month the month.
543  * @param yyyy the year (in the range 1900 to 9999).
544  *
545  * @return the number of the last day of the month.
546  */
547 public static int lastDayOfMonth(final int month, final int yyyy) {
548
549     final int result = LAST_DAY_OF_MONTH[month];
550     if (month != FEBRUARY) {
551         return result;
552     }
553     else if (isLeapYear(yyyy)) {
554         return result + 1;
555     }
556     else {
557         return result;
558     }
559 }
560
561 /**
562  * Creates a new date by adding the specified number of days to the base
563  * date.
564  *
565  * @param days the number of days to add (can be negative).
566  * @param base the base date.
567  *
568  * @return a new date.
569  */
570 public static SerialDate addDays(final int days, final SerialDate base) {
571
572     final int serialDayNumber = base.toSerial() + days;
573     return SerialDate.createInstance(serialDayNumber);
574 }
575
576 /**
577  * Creates a new date by adding the specified number of months to the base
578  * date.
579  * <P>
580  * If the base date is close to the end of the month, the day on the result
581  * may be adjusted slightly: 31 May + 1 month = 30 June.
582  *
583  * @param months the number of months to add (can be negative).
584  * @param base the base date.
585  *
586  * @return a new date.
587  */
588 public static SerialDate addMonths(final int months,
589                                     final SerialDate base) {
590
591     final int yy = (12 * base.getYYYY() + base.getMonth() + months - 1)
592                   / 12;
593     final int mm = (12 * base.getYYYY() + base.getMonth() + months - 1)
594                   % 12 + 1;
595     final int dd = Math.min(
596         base.getDayOfMonth(), SerialDate.lastDayOfMonth(mm, yy)
597     );
598 }

```



### B.1. példa

#### SerialDate.java (folytatás)

```

599     );
600     return SerialDate.createInstance(dd, mm, yy);
601
602 }
603
604 /**
605  * Creates a new date by adding the specified number of years to the base
606  * date.
607  *
608  * @param years the number of years to add (can be negative).
609  * @param base the base date.
610  *
611  * @return A new date.
612  */
613 public static SerialDate addYears(final int years, final SerialDate base) {
614     final int baseY = base.getYYYY();
615     final int baseM = base.getMonth();
616     final int baseD = base.getDayOfMonth();
617
618     final int targetY = baseY + years;
619     final int targetD = Math.min(
620         baseD, SerialDate.lastDayOfMonth(baseM, targetY)
621     );
622
623     return SerialDate.createInstance(targetD, baseM, targetY);
624 }
625
626 /**
627  * Returns the latest date that falls on the specified day-of-the-week and
628  * is BEFORE the base date.
629  *
630  * @param targetWeekday a code for the target day-of-the-week.
631  * @param base the base date.
632  *
633  * @return the latest date that falls on the specified day-of-the-week and
634  *         is BEFORE the base date.
635  */
636 public static SerialDate getPreviousDayOfWeek(final int targetWeekday,
637     final SerialDate base) {
638     // check arguments...
639     if (!SerialDate.isValidWeekdayCode(targetWeekday)) {
640         throw new IllegalArgumentException(
641             "Invalid day-of-the-week code."
642         );
643     }
644
645     // find the date...
646     final int adjust;
647     final int baseDOW = base.getDayOfWeek();
648     if (baseDOW > targetWeekday) {
649         adjust = Math.min(0, targetWeekday - baseDOW);
650     }
651     else {
652         adjust = -7 + Math.max(0, targetWeekday - baseDOW);
653     }
654
655     return SerialDate.addDays(adjust, base);
656 }
657
658 }
659
660

```