10/26/15 01:55:06 /home/15504319/DSA120/DSAAssignment/connorLib/DateClass.java

```
  1  /*******************************************************************************
  2  *  FILE: DateClass.java
  3  *  AUTHOR: Connor Beardsmore - 15504319
  4  *  UNIT: DSA120 Assignment S2- 2015
  5  *  PURPOSE: Container class to represent a date as 3 seperate integer values
  6  *  LAST MOD: 19/10/15
  7  *  REQUIRES: NONE
  8  *******************************************************************************/
  9  package connorLib;
 10  import java.lang.*;
 11
 12  public class DateClass implements Comparable<DateClass>
 13  {
 14      //CLASSFIELDS
 15      private static final int[] DAYS = { 0, 31, 29, 31, 30, 31, 30, 31, 31,
 16                                                          30, 31, 30, 31 };
 17      //Once we set values, the date does not change
 18      private final int year; // 0000 to 9999 valid
 19      private final int month; // 0 to 12 valid
 20      private final int day;  // 0 to DAYS[month] valid

 21  //--------------------------------------------------------------------------
 22      //ALTERNATE Constructor
 23      //IMPORT: inYear (int), inMonth (int), inDay (int)
 24
 25      public DateClass(int inYear, int inMonth, int inDay)
 26      {
 27          //Validation utilizing DAYS constant array
 28          if ( inMonth > (DAYS.length - 1) )
 29          {
 30              throw new IllegalArgumentException("Invalid Date (Month)");
 31          }
 32          if ( !isValid( inYear, inMonth, inDay ) )
 33          {
 34              throw new IllegalArgumentException("Invalid Date");
 35          }
 36          year = inYear;
 37          month = inMonth;
 38          day = inDay;
 39      }
 40  //--------------------------------------------------------------------------
 41      //ALTERNATE Constructor
 42      //IMPORT: date (String)
 43      //PURPOSE: Construct date from string of format YYYY-MM-DD
 44
 45      public DateClass(String date)
 46      {
 47          String[] fields = date.split("-");
 48          if ( fields.length != 3 )
 49          {
 50              throw new IllegalArgumentException("Invalid Date (Too Long)");
 51          }
 52
 53          try
 54          {
 55              year = Integer.parseInt( fields[0] );
 56              month = Integer.parseInt( fields[1] );
 57              day = Integer.parseInt( fields[2] );
 58          }
 59          catch (NumberFormatException e)
 60          {
 61              throw new NumberFormatException("Date Can Only Contain Numbers");
 62          }

 63
 64
 65          if ( month > (DAYS.length - 1) )
 66          {
 67              throw new IllegalArgumentException("Invalid Date (Month)");
 68          }
 69          if ( !isValid( year, month, day ) )
 70          {
 71              throw new IllegalArgumentException("Invalid Date");
 72          }
 73      }
 74  //--------------------------------------------------------------------------
 75      //ACCESSOR getYear
 76      //EXPORT: year (int)
 77
 78      public int getYear()
 79      {
 80          return year;
 81      }
 82  //--------------------------------------------------------------------------
 83      //ACCESSOR getMonth
```

```
84      //EXPORT: month (int)
85
86      public int getMonth()
87      {
88          return month;
89      }
90  //-------------------------------------------------------------------------
91      //ACCESSOR getDay
92      //EXPORT: day (int)
93
94      public int getDay()
95      {
96          return day;
97      }
98  //-------------------------------------------------------------------------
99      //ACCESSOR isValid
100     //IMPORT: inYear (int), inMonth (int), inDay (int)
101     //EXPORT: valid (boolean)
102
103     private static boolean isValid(int inYear, int inMonth, int inDay)
104     {
105         boolean valid = true;
106
107         if ( (inMonth < 1) || (inMonth > 12) )
108         {
109             valid = false;
110         }
111         if ( (inDay < 1) || (inDay > DAYS[inMonth]) )
112         {
113             valid = false;
114         }
115         if ( (inMonth == 2) && (inDay == 29) && (!isLeapYear(inYear)) )
116         {
117             valid = false;
118         }
119         if ( (inYear < 1) || (inYear > 9999) )
120         {
121             valid = false;
122         }
123         //Accounts for default date values of 0000-00-00 format
124         if ( (inYear == 0) && (inMonth == 0) && (inDay == 0) )
125         {
126             valid = true;
127         }
128
129         return valid;
130     }
131 //-------------------------------------------------------------------------
132     //withinMonths
133     //IMPORT: inDate (DateClass), urgency (int)
134     //EXPORT: isWithin (boolean)
135     //PURPOSE: Checks if inDate is within 'urgency' months of this date
136     //           Returns TRUE if it is, FALSE if not
137
138     public boolean withinMonths(DateClass inDate, int urgency)
139     {
140         boolean isWithin = false;
141
142         if ( year == inDate.getYear() )
143         {
144             if ( ( month - inDate.getMonth() ) < urgency )
145             {
146                 isWithin = true;
147             }
148         }
149         return isWithin;
150     }
151 //-------------------------------------------------------------------------
152     //compareTo
153     //IMPORT: inDate (DateClass)
154     //PURPOSE: Compares 2 dates for use in sorting. 0 = dates equals
155     //           -ve = this.date is less than. +ve = this.date is more than
156     //         OVERRIDES compareTo in Comparable
157
158     public int compareTo(DateClass inDate)
159     {
160         int comparison = 0;
161
162         if ( isInfinite() )
163         {
164             comparison = +1;
165         }
166         else if ( year < inDate.getYear() )
167         {
168             comparison = -1;
169         }
170         else if ( year > inDate.getYear() )
```

```
171         {
172             comparison = +1;
173         }
174         else if ( month < inDate.getMonth() )
175         {
176             comparison = -1;
177         }
178         else if ( month > inDate.getMonth() )
179         {
180             comparison = +1;
181         }
182         else if ( day < inDate.getDay() )
183         {
184             comparison = -1;
185         }
186         else if ( day > inDate.getDay() )
187         {
188             comparison = +1;
189         }
190         return comparison;
191     }
192 //-------------------------------------------------------------------------
193     //ACCESSOR isLeapYear
194     //IMPORT: inYear (int)
195     //EXPORT: leapYear (boolean)
196 
197     private static boolean isLeapYear(int inYear)
198     {
199         return ( (inYear % 100 == 0) && (inYear % 4 == 0)
200                                      || (inYear % 400 == 0) );
201     }
202 //-------------------------------------------------------------------------
203     //ACCESSOR isInfinite
204     //EXPORT: infinite (boolean)
205     //PURPOSE: Check if date is unspecified/infinite/default. i.e. 0000-00-00
206 
207     public boolean isInfinite()
208     {
209         return ( (year == 0) && (month == 0) && (day == 0) );
210     }
211 //-------------------------------------------------------------------------
212     //ACCESSOR toString
213     //EXPORT: statestring (String)
214     //PURPOSE: Exports string in format YYYY-MM-DD
215 
216     public String toString()
217     {
218         return String.format("%04d-%02d-%02d", year, month, day);
219     }
220 //-------------------------------------------------------------------------
221 }
```