```java
package project1;

import java.io.*;
import java.util.InputMismatchException;
import java.util.Scanner;

/**********************************************************************
 * This program builds a GUI date clock and counts up to a desired date.
 * Multiple functions are used to count the clock
 *
 * @author Nicholas Layman
 * @author Justin Von Kulajta Winn
 * @version 1.8
 *********************************************************************/

public class GeoCountDownTimer {

    /** This is the number of months for a given date */
    private int month;

    /** this is the number of years for a given date */
    private int year;

    /** this is the number of days for a given date */
    private int day;

    /** this is the minimum year the count down timer can go */
    private static final int MINYEAR = 2019;

    /**********************************************************************
     * This is a basic constructor and has no inputs or returns. This sets
     * the date to the earliest date possible
     *********************************************************************/
    public GeoCountDownTimer() {
        this.month = 1;
        this.day = 1;
        this.year = MINYEAR;
    }

    /**********************************************************************
     * This is a constructor that sets the month, day, and year so long as
     * the given inputs make a valid date
     * @param pMonth is the month to be set as the instance variable
     * @param pDay is the day to be set as the instance variable
     * @param pYear is the year to be set as the instance variable
     * @throws IllegalArgumentException is thrown when the date is invalid
     *********************************************************************/
    public GeoCountDownTimer(int pMonth, int pDay, int pYear){
        if (!isValidDate(pMonth, pDay, pYear))
            throw new IllegalArgumentException();

        this.month = pMonth;
        this.day = pDay;
        this.year = pYear;
    }

    /**********************************************************************
     * This is a constructor that sets the instance variables month, day,
     * and year to that of another, given GeoCountDownTimer
     * @param pOther is the GeoCountDownTimer being used to set the instance variables
     *********************************************************************/
    public void GeoCountDownTimer(GeoCountDownTimer pOther){
        this.month = pOther.month;
        this.day = pOther.day;
        this.year = pOther.year;
    }

    /**********************************************************************
     * This is a constructor that breaks up a string and sets the instance
     * variables equal to the respective component
     * @param geoDate is the string that holds the month, day and year
     *                in the format "mm/dd/yyyy"
     * @throws IllegalArgumentException is thrown when the given date
     *                                  is invalid. It is also used when the
     *                                  string is formatted incorrectly
```

```
 76      ********************************************************************/
 77     public GeoCountDownTimer(String geoDate){
 78         String[] date = geoDate.split("/", 0);
 79 
 80         if (date.length == 3){
 81             this.month = Integer.parseInt(date[0]);
 82             this.day = Integer.parseInt(date[1]);
 83             this.year = Integer.parseInt(date[2]);
 84         } else
 85             throw new IllegalArgumentException();
 86 
 87         if (!isValidDate(this.month, this.day, this.year))
 88             throw new IllegalArgumentException();
 89     }
 90 
 91     /**********************************************************************
 92      * This function checks if the GeoCountDownTimer is equal to the object
 93      * sent in
 94      * @param pOther is an object that is likely to be holding a GeoCountDownTimer
 95      * @throws IllegalArgumentException when the object is not holding anything
 96      *                                  or if the object is not a GeoCountDownTimer
 97      * @return true if the month, day, and year match. returns false if they do not
 98      *********************************************************************/
 99     public boolean equals(Object pOther){
100         if (pOther != null) {
101             if (pOther instanceof GeoCountDownTimer) {
102                 GeoCountDownTimer temp = (GeoCountDownTimer) pOther;
103                 return this.year == temp.year && this.month == temp.month
104                         && this.day == temp.day;
105             }
106         }
107         throw new IllegalArgumentException();
108     }
109 
110     /**********************************************************************
111      *  This function compares two GeoCountDownTimer variables
112      * @param other is the 2nd GeoCountDownTimer that is being compared to the
113      *              instance variables
114      * @return 1 if 'this' is a later date than 'other', 0 if they are the same date,
115      *         and -1 if 'other' is a later date 'this'
116      *********************************************************************/
117     public int compareTo(GeoCountDownTimer other){
118         if (this.equals(other))
119             return 0;
120 
121         if (this.year < other.year)
122             return -1;
123 
124         if (this.year == other.year && this.month < other.month)
125             return -1;
126 
127         if (this.year == other.year && this.month == other.month && this.day < other.day)
128             return -1;
129 
130         return 1;
131     }
132 
133     /**********************************************************************
134      * This function decreases the date by 1 day
135      *********************************************************************/
136     public void dec(){
137         this.dec(1);
138     }
139 
140     /**********************************************************************
141      * This function increases the date by 1 day
142      *********************************************************************/
143     public void inc(){
144         this.inc(1);
145     }
146 
147     /** this is the array for days in each month */
148     private static final int[] DAYS_IN_MONTH = {0, 31, 28, 31, 30, 31, 30, 31,
149             31, 30, 31, 30, 31};
150 
```

```
151     /** this is the array that holds the names of each month */
152     private static final String[] MONTHS = {"", "January", "February",
153             "March", "April", "May", "June", "July", "August", "September",
154             "October", "November", "December"};
155
156     /**************************************************************************
157      * This function takes in a month and year and returns how many days are
158      * in that month. Useful for checking leap years.
159      * @param pMonth is the month being checked
160      * @param pYear is the year being checked in
161      * @return the number of days in the given month
162      * @throws IllegalArgumentException when the month or year is out of range
163      *************************************************************************/
164     public static int daysInMonth(int pMonth, int pYear){
165         if (pMonth < 1 || pMonth > 12)
166             throw new IllegalArgumentException();
167
168         if (!isLeapYear(pYear) || pMonth != 2)
169             return DAYS_IN_MONTH[pMonth];
170         else
171             return 29;
172     }
173
174     /**************************************************************************
175      *This function takes in a month, day, and year and returns if that
176      * date is a valid date, if it could ever happen
177      * @param month is the given month
178      * @param day is the given day
179      * @param year is the given year
180      * @return false if any of the values are out of range and true otherwise
181      *************************************************************************/
182     public static boolean isValidDate(int month, int day, int year) {
183         if (month < 1 || month > 12)
184             return false;
185
186         if (day < 1 || day > daysInMonth(month, year))
187             return false;
188
189         if (year < MINYEAR)
190             return false;
191
192         return true;
193     }
194
195     /**************************************************************************
196      * This function determines if the current year is a leap year or not
197      * @param year is the year being tested whether or not it is a leap year
198      * @return true if it is a leap year, false if it is not
199      *************************************************************************/
200     public static boolean isLeapYear(int year) {
201         return year % 4 == 0 && (year % 100 != 0 || year % 400 == 0);
202     }
203
204     /**************************************************************************
205      * This function checks if the 2 given GeoCountDownTimers are equal
206      * @param s1 is the first GeoCountDownTimer
207      * @param s2 is the second GeoCountDownTimer
208      * @return true if the month, day, and year are each equal and false otherwise
209      * @throws IllegalArgumentException if either object is null or is not
210      *                                  a GeoCountDownTimer
211      *************************************************************************/
212     public static boolean equals(Object s1, Object s2) {
213         if (s1 != null && s2 != null) {
214             if (s1 instanceof GeoCountDownTimer && s2 instanceof GeoCountDownTimer) {
215                 GeoCountDownTimer temp1 = (GeoCountDownTimer) s1;
216                 GeoCountDownTimer temp2 = (GeoCountDownTimer) s2;
217                 return temp1.year == temp2.year && temp1.month == temp2.month
218                         && temp1.day == temp2.day;
219             }
220         }
221         throw new IllegalArgumentException();
222     }
223
224     /**************************************************************************
225      * This function returns a string of the date in the form "Month_name day, year"
```

```
226      * @return the string holding the month, day, and year in the proper form
227      *************************************************************************/
228     public String toString() {
229         return MONTHS[this.month] + " " + this.day + ", " + this.year;
230     }
231 
232     /*************************************************************************
233      * This function returns a string of the date in the form month/day/year
234      * @return a string holding the month, day and year in the specified form
235      *************************************************************************/
236     public String toDateString(){
237         return this.month + "/" + this.day + "/" + this.year;
238     }
239 
240     /*************************************************************************
241      * This function decreases the date by pDays
242      * @param pDays is the number of days the date is being decreased by
243      * @throws IllegalArgumentException when pDays is negative or the year falls
244      *                                  below the MINYEAR
245      *************************************************************************/
246     public void dec(int pDays) {
247         if (pDays < 0)
248             throw new IllegalArgumentException();
249 
250         this.day -= pDays;
251 
252         // keep adding the number of days in the month while moving the
253         // month back until day is positive
254         while (this.day < 1){
255             this.month -= 1;
256 
257             if (this.month < 1){
258                 this.month += 12;
259                 this.year -= 1;
260             }
261 
262             this.day += daysInMonth(this.month, this.year);
263 
264             if (this.year < MINYEAR){
265                 throw new IllegalArgumentException();
266             }
267         }
268     }
269 
270     /*************************************************************************
271      *This function increases the date by pDays
272      * @param pDays is the number of days the date is being increased by
273      * @throws IllegalArgumentException when pDays is negative
274      *************************************************************************/
275     public void inc(int pDays) {
276         if (pDays < 0)
277             throw new IllegalArgumentException();
278 
279         this.day += pDays;
280 
281         // keep subtracting the number of days in the month off while moving
282         // the month forward until day is a valid number
283         while (this.day > daysInMonth(this.month, this.year)){
284             this.day -= daysInMonth(this.month, this.year);
285             this.month += 1;
286 
287             if (this.month > 12){
288                 this.month -= 12;
289                 this.year += 1;
290             }
291         }
292     }
293 
294     /*************************************************************************
295      * This function saves the current date in the given file name
296      * @param filename is the name of the file the data is being saved into
297      * @throws IllegalArgumentException if filename is invalid
298      *************************************************************************/
299     public void save(String filename) {
300 
```

```
301         PrintWriter out = null;
302         try {
303             out = new PrintWriter(new BufferedWriter(new FileWriter(filename)));
304         } catch (IOException e) {
305             throw new IllegalArgumentException();
306         }
307 
308         out.println (month);
309         out.println (day);
310         out.println (year);
311 
312         out.close();
313     }
314 
315     /**************************************************************************
316      * This function pulls up the saved month, day, and year and sets 'this'
317      * equal to them
318      * @param filename is the name of the file the date is being pulled from
319      * @throws IllegalArgumentException if the file does not exist or if its
320      *                                  contents are not in the necessary format
321      *************************************************************************/
322     public void load(String filename){
323         try{
324             Scanner fileReader = new Scanner(new File(filename));
325 
326             this.month = fileReader.nextInt();
327             this.day = fileReader.nextInt();
328             this.year = fileReader.nextInt();
329         } catch (Exception error) {
330             throw new IllegalArgumentException();
331         }
332     }
333 
334     /**************************************************************************
335      * This function finds the number of days to go until timer reaches 0
336      * @param fromDate is the string holding the desired date to check from.
337      *                 This would normally be today.
338      * @return the numbers of days between the two dates
339      * @throws IllegalArgumentException if the date passed is later than 'this'
340      *************************************************************************/
341     public int daysToGo(String fromDate){
342         GeoCountDownTimer temp = new GeoCountDownTimer(fromDate);
343 
344         if (this.compareTo(temp) < 0)
345             throw new IllegalArgumentException();
346 
347         int daysToGo = 0;
348 
349         // keep moving the fromdate timer forward by 1 until
350         // the other timer is reached
351         while (this.compareTo(temp) > 0){
352             daysToGo += 1;
353             temp.inc();
354         }
355 
356         return daysToGo;
357     }
358 
359     /**************************************************************************
360      * This date finds the date which is n days from 'this'.
361      * @param n is the number of days the found date should be from 'this'
362      *          can be positive or negative
363      * @return GeoCountDownTimer temp which is the future/past date found
364      *************************************************************************/
365     public GeoCountDownTimer daysInFuture(int n){
366         GeoCountDownTimer temp = new GeoCountDownTimer(this.toDateString());
367 
368         if (n > 0)
369             temp.inc(n);
370         else
371             temp.dec(-n);
372 
373         return temp;
374     }
375 
```

```
376     /**************************************************************************
377      * This is the main function where tests are being run
378      *************************************************************************/
379     public static void main() {
380         GeoCountDownTimer s = new GeoCountDownTimer("2/10/2020");
381         System.out.println("Date: " + s);
382 
383         GeoCountDownTimer s1 = new GeoCountDownTimer("2/10/2022");
384         System.out.println("Date: " + s1.toString());
385 
386         s1.dec(365);
387         System.out.println("Date: " + s1);
388 
389         GeoCountDownTimer s2 = new GeoCountDownTimer("2/10/2019");
390         for (int i = 0; i < (365 + 366); i++)
391             s2.inc(1);
392         System.out.println("Date: " + s2);
393 
394 
395         GeoCountDownTimer s3 =  new GeoCountDownTimer(5, 5, 2019);
396         s3.dec(700);
397         System.out.println("Date: " + s3);
398 
399         s3.inc(1000);
400         System.out.println("Date: " + s3.toDateString());
401 
402         GeoCountDownTimer s4 = new GeoCountDownTimer("11/7/2019");
403         GeoCountDownTimer s5 = new GeoCountDownTimer();
404         s5.GeoCountDownTimer(s4);
405 
406         try{
407             s4.dec(3660);
408             s5.dec(3660);
409             System.out.println("FAILED");
410         } catch (IllegalArgumentException e){
411             System.out.println("Date: " + s4.toDateString());
412             System.out.println("Date: " + s5.toDateString());
413         }
414 
415 
416         System.out.println("Comparison Tests");
417         System.out.println("If the first timer is greater than the other timer, then the test");
418         System.out.println("returns a 1");
419         System.out.println("If the two timers are equal, the test will return a 0");
420         System.out.println("If the second timer is greater than the first, then the test");
421         System.out.println("returns a -1");
422         System.out.println("Comparison Value: " + s4.compareTo(s5));
423 
424         System.out.println("Save/Load Test");
425         GeoCountDownTimer s6 = new GeoCountDownTimer("1/1/2018");
426 
427         s6.inc();
428         s6.save("temp.txt");
429 
430         s6.load("temp.txt");
431 
432         System.out.println("Loaded Date: "+ s6);
433 
434         System.out.println("Days To Go Test!");
435 
436         System.out.println("Days between s6 and '12/12/2017:' " + s6.daysToGo("12/12/2017"));
437 
438         System.out.println("Days in the Future Test");
439 
440         GeoCountDownTimer s7 = new GeoCountDownTimer();
441         s7.GeoCountDownTimer(s6.daysInFuture(5));
442         System.out.println("Future date should be '1/7/2018");
443         System.out.println("Actual date: " + s7);
444     }
445 }
```