92/100

89/100

Project 1: "StopWatch" Program Rubric.

Student Name Aaron Teague	
Due Date	
Date Submitted, Days Late, Late Penalty	

Graded Item	Points	Comments and Points Secured
Javadoc Comments and Coding Style/Technique (http://www.cis.gvsu.edu/studentsupport/javaguide) • Code Indentation (auto format source code in IDE) • Naming Conventions (see Java style guide) • Proper access modifiers for fields and methods • Use of helper (private) methods • Using good variable names • Header/class comments • Every method uses @param and @return • Every method uses a /********** separator • Overall layout, readability, No text wrap • Using /** … / for each Instance variable • Has many inner "inner" comments	10	✓ -2
Steps 1 – 2: Basic Functionality • `public StopWatch()` • `public StopWatch(int minutes, int seconds, int milliseconds)` • `public StopWatch(int seconds, int milliseconds)` • `public StopWatch(int milliseconds)` • `public StopWatch(String startTime)` • `public boolean equals(StopWatch other)` • `public static boolean equals(StopWatch s1, StopWatch s2)` • `public int compareTo(StopWatch other)` • `public void add(int milliseconds` • `public void add(StopWatch other)` • `public void inc()` • `public String toString()` • `etc….`	40	
Step 3: JUnit test	10	97/97 ✓
Step 4: Main test	5	-1
Step 5: Added functionality • public void save(String fileName) • public void load(String fileName) • public static setMutate (boolean …) • Allow for an error in the input for all constructors and methods	20	
Step 6: More Software Testing	5	

SEE NEXT PAGE

Step 7: Challenge Requirement	10	ONE TIMER -1 7
Total	**100**	

Additional Comments:

TIMER DOESÚT WORK IN GUI
WORKS IN TESTS THOUGH

8

```java
/**********************************************************************
Simulation of a StopWatch

@author Aaron Teague
 **********************************************************************/

package package1;

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

public class StopWatch {

    /** Boolean value that control if the StopWatch object can change */
    private static boolean mutate = true;

    /** Number of minutes */
    private int minutes;

    /** Number of seconds */
    private int seconds;

    /** Number of milliseconds */
    private int milliseconds;

    /**********************************************************************
     * Returns the number minutes.
     *
     * @return minutes
     **********************************************************************/
    public int getMinutes() {
        return this.minutes;
    }

    /**********************************************************************
     * Sets the number minutes.
     *
     * @param minutes
     **********************************************************************/
    public void setMinutes(int minutes)throws IllegalArgumentException {
        if (mutate) {
            if (minutes >= 0)
                this.minutes = minutes;
            else
                throw new IllegalArgumentException();
        }
    }

    /**********************************************************************
     * Returns the number of seconds.
     *
     * @return seconds
     **********************************************************************/
    public int getSeconds() {
        return this.seconds;
```

```
56      }
57
58      /**********************************************************************
59       * Sets the number of seconds.
60       *
61       * @param seconds
62       **********************************************************************/
63      public void setSeconds(int seconds)throws IllegalArgumentException {
64          if (mutate) {
65              if (seconds >= 0 && seconds < 60)
66                  this.seconds = seconds;
67              else
68                  throw new IllegalArgumentException();
69          }
70      }
71
72      /**********************************************************************
73       * Returns the number of milliseconds.
74       *
75       * @returns milliseconds
76       **********************************************************************/
77      public int getMilliseconds() {
78          return this.milliseconds;
79      }
80
81      /**********************************************************************
82       * Sets the number of milliseconds
83       *
84       * @param milliseconds
85       **********************************************************************/
86      public void setMilliseconds(int milliseconds)
87              throws IllegalArgumentException {
88          if (mutate) {
89              if (milliseconds >= 0 && milliseconds < 1000)
90                  this.milliseconds = milliseconds;
91              else
92                  throw new IllegalArgumentException();
93          }
94      }
95
96      /**********************************************************************
97       * Default constructor, creates a Stop Watch with 0 minutes,
98       * 0 seconds, and 0 milliseconds
99       **********************************************************************/
100     public StopWatch() {
101
102         this.minutes = 0;
103         this.seconds = 0;
104         this.milliseconds = 0;
105
106     }
107
108     /**********************************************************************
109      * Constructor, creates a Stop Watch: sets the minutes, the seconds,
110      * and the milliseconds
```

```
111      *
112      * @param minutes
113      * @param seconds
114      * @param milliseconds
115      ***********************************************************************/
116     public StopWatch(int minutes, int seconds, int milliseconds)
117             throws IllegalArgumentException {
118
119         if (seconds < 60 && milliseconds < 1000) {
120
121             if (minutes >= 0 && seconds >= 0 && milliseconds >= 0) {
122
123                 this.minutes = minutes;
124                 this.seconds = seconds;
125                 this.milliseconds = milliseconds;
126             } else
127                 throw new IllegalArgumentException();
128         } else
129             throw new IllegalArgumentException();
130
131     }
132
133     /***********************************************************************
134      * Constructor, creates a Stop Watch: sets the seconds, and
135      * milliseconds
136      *
137      * @param seconds
138      * @param milliseconds
139      ***********************************************************************/
140     public StopWatch(int seconds, int milliseconds)
141             throws IllegalArgumentException {
142
143         if (seconds < 60 && milliseconds < 1000) {
144
145             if (seconds >= 0 && milliseconds >= 0) {
146
147                 this.seconds = seconds;
148                 this.milliseconds = milliseconds;
149             }
150
151             else
152
153                 throw new IllegalArgumentException();
154         }
155
156         else
157             throw new IllegalArgumentException();
158     }
159
160     /***********************************************************************
161      * Constructor, creates a Stop Watch: sets the milliseconds
162      *
163      * @param milliseconds
164      ***********************************************************************/
165     public StopWatch(int milliseconds) throws IllegalArgumentException {
```

```
166
167         if (milliseconds >= 0 && milliseconds < 1000)
168             this.milliseconds = milliseconds;
169
170         else
171             throw new IllegalArgumentException();
172     }
173
174     /**********************************************************************
175      * Constructor, creates a Stop Watch from a string
176      *
177      * @param startTime, the time in a string
178      *********************************************************************/
179     public StopWatch(String startTime) throws IllegalArgumentException {
180
181         String[] parts = startTime.split(":");
182
183         int i = parts.length;
184
185         switch (i) {
186
187         case 3: {
188
189             if (Integer.parseInt(parts[1]) < 60
190                     && Integer.parseInt(parts[2]) < 1000) {
191
192                 if (Integer.parseInt(parts[0]) >= 0
193                         && Integer.parseInt(parts[1]) >= 0
194                         && Integer.parseInt(parts[2]) >= 0) {
195
196                     this.minutes = Integer.parseInt(parts[0]);
197                     this.seconds = Integer.parseInt(parts[1]);
198                     this.milliseconds = Integer.parseInt(parts[2]);
199                     break;
200                 }
201
202                 else
203                     throw new IllegalArgumentException();
204             }
205
206             else
207                 throw new IllegalArgumentException();
208         }
209
210         case 2: {
211
212             if (Integer.parseInt(parts[0]) < 60
213                     && Integer.parseInt(parts[1]) < 1000) {
214
215                 if (Integer.parseInt(parts[0]) >= 0
216                         && Integer.parseInt(parts[1]) >= 0) {
217
218                     this.minutes = 0;
219                     this.seconds = Integer.parseInt(parts[0]);
220                     this.milliseconds = Integer.parseInt(parts[1]);
```

```
221                     break;
222                 }
223
224                 else
225                     throw new IllegalArgumentException();
226             }
227
228             else
229                 throw new IllegalArgumentException();
230         }
231
232         case 1: {
233
234             if (Integer.parseInt(startTime) >= 0
235                     && Integer.parseInt(startTime) < 1000) {
236                 this.minutes = 0;
237                 this.seconds = 0;
238                 this.milliseconds = Integer.parseInt(startTime);
239                 break;
240             }
241
242             else
243                 throw new IllegalArgumentException();
244
245             }
246         }
247
248     }
249
250     /*********************************************************************
251      * Boolean method takes a StopWatch argument. Returns true if the
252      * current StopWatch object is exactly the same as the StopWatch
253      * passed in
254      *
255      * @param other, StopWatch object
256      * @return boolean true or false
257      ********************************************************************/
258     public boolean equals(StopWatch other) {
259
260         if (this.minutes == other.minutes
261                 && this.seconds == other.seconds
262                 && this.milliseconds == other.milliseconds)
263
264             return true;
265
266         else
267
268             return false;
269     }
270
271     /*********************************************************************
272      * Boolean method that is able to accept a Object argument. Returns
273      * true if the current StopWatch object is exactly the same as the
274      * object passed in
275      *
```

```
276      * @param other, a Object type object
277      * @return boolean true or false
278      ***************************************************************/
279     public boolean equals(Object other) {
280
281         if (this.minutes == ((StopWatch) other).minutes
282             && this.seconds == ((StopWatch) other).seconds
283             && this.milliseconds == ((StopWatch) other).milliseconds)
284
285             return true;
286
287         else
288
289             return false;
290     }
291
292     /***************************************************************
293      * Static method of a boolean type and accepts two StopWatch object
294      * arguments. Returns true if the first StopWatch is exactly the
295      * same as the second StopWatch
296      *
297      * @param s1, s2: Stopwatch objects
298      * @return boolean true or false
299      ***************************************************************/
300     public static boolean equals(StopWatch s1, StopWatch s2) {
301
302         if (s1.minutes == s2.minutes && s1.seconds == s2.seconds
303                 && s1.milliseconds == s2.milliseconds)
304
305             return true;
306
307         else
308
309             return false;
310     }
311
312     /***************************************************************
313      * Method returns "1" if the current StopWatch is greater than the
314      * StopWatch passed in; returns "-1" if the current StopWatch is
315      * less than the StopWatch passed in; returns "0" if the current
316      * StopWatch is equal to theStopWatch passed in
317      *
318      * @param other, StopWatch object
319      * @return 0, 1, -1
320      ***************************************************************/
321     public int compareTo(StopWatch other) {
322
323         if (this.equals(other))
324             return 0;
325
326         if (this.minutes > other.minutes)
327             return 1;
328         else if (this.seconds > other.seconds)
329             return 1;
330         else if (this.milliseconds > other.milliseconds)
```

```
331             return 1;
332         else
333             return -1;
334
335     }
336
337     /*************************************************************
338      * Method adds milliseconds to the current StopWatch
339      *
340      * @param milliseconds
341      * @return none
342      *************************************************************/
343     public void add(int milliseconds) {
344       if(mutate){
345         if (milliseconds >= 0) {
346             this.minutes += milliseconds / 60000;
347             milliseconds %= 60000;
348             this.seconds += milliseconds / 1000;
349
350             if (this.seconds >= 60) {
351                 this.minutes += this.seconds / 60;
352                 this.seconds %= 60;
353             }
354
355             milliseconds = milliseconds % 1000;
356
357             while (milliseconds > 0) {
358                 this.inc();
359                 milliseconds--;
360
361             }
362         }
363       }
364     }
365
366     /*********************************************************************
367      * Method adds StopWatch time passed in to the current StopWatch
368      *
369      * @param other, StopWatch object
370      * @return none
371      *********************************************************************/
372     public void add(StopWatch other) {
373
374         if (mutate) {
375             int total = 0;
376             total = other.minutes * 60000 + other.seconds * 1000
377                     + other.milliseconds;
378
379             this.add(total);
380         }
381
382     }
383
384     /*********************************************************************
385      * Method increments the milliseconds of the current StopWatch
```

```
386      *
387      * @param none
388      * @return none
389      *********************************************************************/
390     public void inc() {
391
392         if (mutate) {
393             this.milliseconds++;
394
395             if (this.milliseconds >= 1000) {
396                 this.seconds += this.milliseconds / 1000;
397                 this.milliseconds %= 1000;
398             }
399
400             if (this.seconds >= 60) {
401                 this.minutes += this.seconds / 60;
402                 this.seconds %= 60;
403             }
404         }
405
406     }
407
408     /*********************************************************************
409      * Method returns a formated string of the current StopWatch's time
410      *
411      * @param none
412      * @return the time in a string
413      *********************************************************************/
414     public String toString() {
415
416         String a = String.format("%02d", this.seconds);
417         String b = String.format("%03d", this.milliseconds);
418         return this.minutes + ":" + a + ":" + b;
419     }
420
421     /*********************************************************************
422      * Takes information from a file and sets the minutes, seconds, and
423      * milliseconds
424      *
425      * @param fileName, the file that contains the time
426      * @return none
427      *********************************************************************/
428     public void load(String fileName) {
429
430         String time;
431
432         try {
433             Scanner fileReader = new Scanner(new File(fileName));
434
435             time = fileReader.next();
436             StopWatch other = new StopWatch(time);
437             this.minutes = other.minutes;
438             this.seconds = other.seconds;
439             this.milliseconds = other.milliseconds;
440
```

```
441             fileReader.close();
442         }
443
444         catch (FileNotFoundException error) {
445             System.out.println("File Not Found");
446         }
447
448     }
449
450     /**************************************************************
451      * Writes the time to a file
452      *
453      * @param fileName, the file saving to
454      * @return none
455      *************************************************************/
456     public void save(String fileName) {
457
458         PrintWriter out = null;
459         try {
460             out = new PrintWriter(new BufferedWriter(new FileWriter(
461                     fileName)));
462         } catch (IOException e) {
463             e.printStackTrace();
464
465         }
466
467         out.println(this.toString());
468         out.close();
469
470     }
471
472     /**************************************************************************
473      * Sets the a boolean value "mutate", which is used to determine if
474      * an object can be changed.
475      *
476      * @param mutate, boolean value
477      * @return none
478      *************************************************************************/
479     public static void setMutate(boolean mutate) {
480         StopWatch.mutate = mutate;
481     }
482
483 }
484
```

```java
package package1;

import static org.junit.Assert.*;

import org.junit.Test;

import org.junit.Test;

import static org.junit.Assert.*;

public class TestStopWatch {

/**
 *
 * The following are simple JUnit test cases... After talking with your
 * instructor, create many, many more that shows that your code
 * is functioning correctly.
 *
 */


    //Tests three int constructor with three positive ints
    @Test
    public void testThreeIntConstructor() {
        StopWatch s = new StopWatch (5,10,300);
        assertEquals(s.toString(),"5:10:300");
    }

    //Tests three int constructor with three 0's
    @Test
    public void testZeroThreeIntConstructor() {
        StopWatch s = new StopWatch (0,0,0);
        assertEquals(s.toString(),"0:00:000");
    }

    //Tests three int constructor with a negative int for the minutes
    @Test (expected = IllegalArgumentException.class)
    public void testNegativeMinThreeIntConstructor(){
        new StopWatch(-1,0,0);
    }

    //Tests three int constructor with a negative int for the seconds
    @Test (expected = IllegalArgumentException.class)
    public void testNegativeSecThreeIntConstructor(){
        new StopWatch(0,-1,0);
    }

    //Tests three int constructor with a negative int for the millisec.
    @Test (expected = IllegalArgumentException.class)
    public void testNegativeMilThreeIntConstructor(){
        new StopWatch(0,0,-1);
    }

    //Tests three int constructor with a negative int for the minutes
    // and seconds
```

```
56      @Test (expected = IllegalArgumentException.class)
57      public void testNegativeSecMilThreeIntConstructor(){
58          new StopWatch(0,-1,-1);
59      }
60
61      //Tests three int constructor with a negative int for the millisec.
62      // and seconds
63      @Test (expected = IllegalArgumentException.class)
64      public void testNegativeMinSecThreeIntConstructor(){
65          new StopWatch(-1,-1, 0);
66      }
67
68      //Tests three int constructor with a negative int for the min. and
69      //millisec.
70      @Test (expected = IllegalArgumentException.class)
71      public void testNegativeMinMilThreeIntConstructor(){
72          new StopWatch(-1,0,-1);
73      }
74
75      //Tests three int constructor with a negative int for the minutes
76      // seconds, and millisec.
77      @Test (expected = IllegalArgumentException.class)
78      public void testNegativMinSecMilThreeIntConstructor(){
79          new StopWatch(-1,-1,-1);
80      }
81
82      //Tests three int constructor with 60 seconds
83      @Test (expected = IllegalArgumentException.class)
84      public void testOverMaxSecThreeIntConstructor(){
85          new StopWatch(0,60,0);
86      }
87
88      //Tests three int constructor with 1000 millisec
89      @Test (expected = IllegalArgumentException.class)
90      public void testOverMaxMilThreeIntConstructor(){
91          new StopWatch(0,0,1000);
92      }
93
94      //Tests three int constructor with 60 seconds and 1000 millisec
95      @Test (expected = IllegalArgumentException.class)
96      public void testOverMaxSecAndMilThreeIntConstructor(){
97          new StopWatch(0,60,1000);
98      }
99
100     //Tests three int constructor with 999 millisec
101     @Test
102     public void testAllowedMilThreeIntConstructor(){
103         StopWatch s = new StopWatch (0,0,999);
104         assertEquals(s.toString(),"0:00:999");
105     }
106
107     //Tests three int constructor with 59 sec
108     @Test
109     public void testAllowedMinThreeIntConstructor(){
110         StopWatch s =  new StopWatch (0,59,0);
```

```
111         assertEquals(s.toString(),"0:59:000");
112     }
113
114     //Tests three int constructor with 59 sec and millisec
115     @Test
116     public void testAllowedMinMilThreeIntConstructor(){
117         StopWatch s =  new StopWatch (0,59,999);
118         assertEquals(s.toString(),"0:59:999");
119     }
120
121     //Tests two int constructor with two 0's
122     @Test
123     public void testZeroTwoIntConstructor(){
124         StopWatch s = new StopWatch (0,0);
125         assertEquals(s.toString(),"0:00:000");
126     }
127
128     //Tests two int constructor with two positive ints
129     @Test
130     public void testTwoIntConstructor(){
131         StopWatch s = new StopWatch (30,300);
132         assertEquals(s.toString(),"0:30:300");
133     }
134
135     //Tests two int constructor with negative seconds
136     @Test (expected = IllegalArgumentException.class)
137     public void testNegativeSecTwoIntConstructor(){
138         new StopWatch(-1,0);
139     }
140
141     //Tests two int constructor with negative millisec
142     @Test (expected = IllegalArgumentException.class)
143     public void testNegativeMilTwoIntConstructor(){
144         new StopWatch(0,-1);
145     }
146
147     //Tests two int constructor with negative seconds and millisec
148     @Test (expected = IllegalArgumentException.class)
149     public void testNegativeSecAndMilTwoIntConstructor(){
150         new StopWatch(-1,-1);
151     }
152
153     //Tests two int constructor with 60 seconds
154     @Test (expected = IllegalArgumentException.class)
155     public void overMaxSecTwoIntConstructor(){
156         new StopWatch(60,0);
157     }
158
159     //Tests two int constructor with 1000 millisec
160     @Test (expected = IllegalArgumentException.class)
161     public void overMaxMilTwoIntConstructor(){
162         new StopWatch(0,1000);
163     }
164
165     //Tests two int constructor with 60 seconds and 1000 millisec
```

```
166     @Test (expected = IllegalArgumentException.class)
167     public void overMaxSecAndMilTwoIntConstructor(){
168         new StopWatch(60,1000);
169     }
170
171     //Tests one int constructor with 0 millisec
172     @Test
173     public void testZeroOneIntConstructor(){
174         StopWatch s = new StopWatch (0);
175         assertEquals(s.toString(),"0:00:000");
176     }
177
178     //Tests one int constructor with positive millisec
179     @Test
180     public void testOneIntConstructor(){
181         StopWatch s = new StopWatch (300);
182         assertEquals(s.toString(),"0:00:300");
183     }
184
185     //Tests one int constructor with negative millisec
186     @Test (expected = IllegalArgumentException.class)
187     public void testNegativeOneIntConstructor(){
188         new StopWatch(-1);
189     }
190
191     //Tests one int constructor with 1000 millisec
192     @Test (expected = IllegalArgumentException.class)
193     public void testOverMaxOneIntConstructor(){
194         new StopWatch(1000);
195     }
196
197     //Tests one int constructor with 999 millisec
198     @Test
199     public void testAllowedMaxOneIntConstructor(){
200         StopWatch s = new StopWatch (999);
201         assertEquals(s.toString(),"0:00:999");
202     }
203
204     //Test String constructor with 0's
205     @Test
206     public void testStringZeroThreeValueConstructor(){
207         StopWatch s = new StopWatch ("0:00:000");
208         assertEquals(s.toString(),"0:00:000");
209     }
210
211     //Test String constructor with positive values
212     @Test
213     public void testStringThreeValueConstructor(){
214         StopWatch s = new StopWatch ("45:37:520");
215         assertEquals(s.toString(),"45:37:520");
216     }
217
218     //Test String constructor with 1000 millisec
219     @Test (expected = IllegalArgumentException.class)
220     public void testOverMaxMilStringThreeValueConstructor(){
```

```
221         new StopWatch ("0:00:1000");
222     }
223
224     //Test String constructor with 999 millisec
225     @Test
226     public void testAllowedMaxMilStringThreeValueConstructor(){
227         StopWatch s = new StopWatch ("0:00:999");
228         assertEquals(s.toString(),"0:00:999");
229     }
230
231     //Test String constructor with 60 seconds
232     @Test (expected = IllegalArgumentException.class)
233     public void testOverMaxSecStringThreeValueConstructor(){
234         new StopWatch ("0:60:000");
235     }
236
237     //Test String constructor with 60 seconds and 1000 milli
238     @Test (expected = IllegalArgumentException.class)
239     public void testOverMaxSecAndMilStringThreeValueConstructor(){
240         new StopWatch ("0:60:1000");
241     }
242
243     //Test String constructor with 59 seconds
244     @Test
245     public void testAllowedMaxSecStringThreeValueConstructor(){
246         StopWatch s = new StopWatch ("0:59:000");
247         assertEquals(s.toString(),"0:59:000");
248     }
249
250     //Test String constructor with negative minutes
251     @Test (expected = IllegalArgumentException.class)
252     public void testNegativeMinStringThreeValueConstructor(){
253         new StopWatch ("-1:00:000");
254
255     }
256
257     //Test String constructor with negative seconds
258     @Test (expected = IllegalArgumentException.class)
259     public void testNegativeSecStringThreeValueConstructor(){
260         new StopWatch ("0:-1:000");
261
262     }
263
264     //Test String constructor with negative millisec
265     @Test (expected = IllegalArgumentException.class)
266     public void testNegativeMilStringThreeValueConstructor(){
267         new StopWatch ("0:00:-1");
268
269     }
270
271     //Test String constructor with neg. min., neg. sec., neg. millisec
272     @Test (expected = IllegalArgumentException.class)
273     public void testNegativeMinSecMilStringThreeValueConstructor(){
274         new StopWatch ("-1:-1:-1");
275
```

```
276     }
277
278     //Test String constructor with neg. sec., neg. millisec
279     @Test (expected = IllegalArgumentException.class)
280     public void testNegativeSecMilStringThreeValueConstructor(){
281         new StopWatch ("0:-1:-1");
282
283     }
284
285     //Test String constructor with neg. min, neg. sec.
286     @Test (expected = IllegalArgumentException.class)
287     public void testNegativeMinSecStringThreeValueConstructor(){
288         new StopWatch ("-1:-1:0");
289     }
290
291     //Test String constructor with neg. min, neg. mil.
292     @Test (expected = IllegalArgumentException.class)
293     public void testNegativeMinMilStringThreeValueConstructor(){
294         new StopWatch ("-1:0:-1");
295     }
296
297     //Test String constructor with 0 min and 0 sec
298     @Test
299     public void testZeroTwoValueStringConstructor(){
300         StopWatch s = new StopWatch ("0:0");
301         assertEquals(s.toString(),"0:00:000");
302     }
303
304     //Test String constructor with positive min and sec
305     @Test
306     public void testTwoValueStringConstructor(){
307         StopWatch s = new StopWatch ("37:150");
308         assertEquals(s.toString(),"0:37:150");
309
310     }
311
312     //Test String constructor with neg min and 0 sec
313     @Test (expected = IllegalArgumentException.class)
314     public void testNegativeSecStringTwoValueConstructor(){
315         new StopWatch ("-1:0");
316
317     }
318
319     //Test String constructor with 0 min and neg sec
320     @Test (expected = IllegalArgumentException.class)
321     public void testNegativeMilStringTwoValueConstructor(){
322         new StopWatch ("0:-1");
323
324     }
325
326     //Test String constructor with neg min and neg sec
327     @Test (expected = IllegalArgumentException.class)
328     public void testNegativeSecMilStringTwoValueConstructor(){
329         new StopWatch ("-1:-1");
330
```

```
331     }
332
333     //Test String constructor with 60 min and 0 sec
334     @Test (expected = IllegalArgumentException.class)
335     public void testOverMaxSexStringTwoValueConstructor(){
336         new StopWatch ("60:0");
337
338     }
339
340     //Test String constructor with 0 min and 1000 sec
341     @Test (expected = IllegalArgumentException.class)
342     public void testOverMaxMilStringTwoValueConstructor(){
343         new StopWatch ("0:1000");
344
345     }
346
347     //Test String constructor with 60 min and 1000 sec
348     @Test (expected = IllegalArgumentException.class)
349     public void testOverMaxSecMilStringTwoValueConstructor(){
350         new StopWatch ("60:1000");
351
352     }
353
354     //Test String constructor with 59 min and 0 sec
355     @Test
356     public void testAllowedMaxSecStringTwoValueConstructor(){
357         StopWatch s = new StopWatch ("59:0");
358         assertEquals(s.toString(),"0:59:000");
359
360     }
361
362     //Test String constructor with 0 min and 999 sec
363     @Test
364     public void testAllowedMaxMilStringTwoValueConstructor(){
365         StopWatch s = new StopWatch ("0:999");
366         assertEquals(s.toString(),"0:00:999");
367
368     }
369
370     //Test String constructor with 59 min and 999 sec
371     @Test
372     public void testAllowedMaxSecMilStringTwoValueConstructor(){
373         StopWatch s = new StopWatch ("59:999");
374         assertEquals(s.toString(),"0:59:999");
375
376     }
377
378     //Test String constructor with 0 millisec
379     @Test
380     public void testMilStringOneValueConstructor(){
381         StopWatch s = new StopWatch ("0");
382         assertEquals(s.toString(),"0:00:000");
383     }
384
385     //Test String constructor with positive millisec
```

```
386     @Test
387     public void testMilStringValueConstructor(){
388         StopWatch s = new StopWatch ("500");
389         assertEquals(s.toString(),"0:00:500");
390     }
391
392     //Test String constructor with neg millisec
393     @Test (expected = IllegalArgumentException.class)
394     public void testNegativeMilStringOneValueConstructor(){
395         new StopWatch ("-1");
396
397     }
398
399     //Test String constructor with 1000 millisec
400     @Test (expected = IllegalArgumentException.class)
401     public void testOverMaxMilStringOneValueConstructor(){
402         new StopWatch ("1000");
403
404     }
405
406     //Test String constructor with 999 millisec
407     @Test
408     public void testAllowedMaxStringOneValueConstructor(){
409         StopWatch s = new StopWatch ("999");
410         assertEquals(s.toString(),"0:00:999");
411     }
412
413     @Test
414     public void testSetMinutes(){
415         StopWatch s = new StopWatch (0,0,0);
416         s.setMinutes(10);
417         assertEquals(s.toString(),"10:00:000");
418         s.setMinutes(0);
419         assertEquals(s.toString(),"0:00:000");
420     }
421
422     //Test setMinutes with negative
423     @Test (expected = IllegalArgumentException.class)
424     public void testNegativeSetMinutes(){
425         StopWatch s = new StopWatch (0,0,0);
426         s.setMinutes(-1);
427     }
428
429     @Test
430     public void testSetSeconds(){
431         StopWatch w = new StopWatch();
432         w.setSeconds(20);
433         assertEquals(w.toString(),"0:20:000");
434         w.setSeconds(0);
435         assertEquals(w.toString(),"0:00:000");
436     }
437
438     //Test setSeconds with negative
439     @Test (expected = IllegalArgumentException.class)
440     public void testNegativeSetSeconds(){
```

```
441         StopWatch s = new StopWatch();
442         s.setSeconds(-1);
443     }
444
445     //Test setSeconds with 60 Seconds
446     @Test (expected = IllegalArgumentException.class)
447     public void testOverMaxSetSeconds(){
448         StopWatch s = new StopWatch();
449         s.setSeconds(60);
450     }
451
452     //Test setSeconds with 59 Seconds
453     @Test
454     public void testAllowedMaxSetSeconds(){
455         StopWatch s = new StopWatch();
456         s.setSeconds(59);
457         assertEquals(s.toString(),"0:59:000");
458     }
459
460     @Test
461     public void testSetMilliseconds(){
462         StopWatch s = new StopWatch();
463         s.setMilliseconds(200);
464         assertEquals(s.toString(),"0:00:200");
465         s.setMilliseconds(0);
466         assertEquals(s.toString(),"0:00:000");
467     }
468
469     //Test setMilliseconds with 1000 Milliseconds
470     @Test (expected = IllegalArgumentException.class)
471     public void testOverMaxSetMilliseconds(){
472         StopWatch c = new StopWatch();
473         c.setMilliseconds(1000);
474     }
475
476     //Test setMilliseconds with 999 Milliseconds
477     @Test
478     public void testAllowedMaxSetMilliseconds(){
479         StopWatch d = new StopWatch();
480         d.setMilliseconds(999);
481         assertEquals(d.toString(),"0:00:999");
482     }
483
484     //Test setMilliseconds with a negative
485     @Test (expected = IllegalArgumentException.class)
486     public void testNegativeSetMilliseconds(){
487         StopWatch e = new StopWatch();
488         e.setMilliseconds(-1);
489     }
490
491     //Adds 1 millisecond to 999 milliseconds
492     @Test
493     public void testMiltoSecAddMillMethod () {
494     StopWatch a = new StopWatch (0,0,999);
495     a.add(1);
```

```
496     assertEquals(a.toString(), "0:01:000");
497     }
498
499     //Adds 1000 millisecond to 59 seconds
500     @Test
501     public void testSectoMinAddMillMethod () {
502         StopWatch s = new StopWatch (0,59,0);
503         s.add(1000);
504         assertEquals(s.toString(), "1:00:000");
505     }
506
507     //Adds 1000 millisecond to 59 seconds
508         @Test
509         public void AddMillMakeOneMinMethod() {
510             StopWatch b = new StopWatch (0,0,0);
511             b.add(60000);
512             assertEquals(b.toString(), "1:00:000");
513         }
514
515     @Test
516     public void testAddMethod () {
517         StopWatch s1 = new StopWatch (5,59,300);
518         s1.add(2000);
519         assertEquals (s1.toString(),"6:01:300");
520
521         s1 = new StopWatch (5,59,300);
522         StopWatch s2 = new StopWatch (2,2,300);
523         s1.add(s2);
524         System.out.println (s1);
525         assertEquals (s1.toString(),"8:01:600");
526
527         for (int i = 0; i < 15000; i++)
528             s1.inc();
529             System.out.println (s1);
530         assertEquals (s1.toString(),"8:16:600");
531     }
532
533     //Adds Object s2(1 millsec) to s1(999 millsec)
534     @Test
535     public void testAddStopWatchOneSecMethod () {
536         StopWatch s1 = new StopWatch(0,0,999);
537         StopWatch s2 = new StopWatch(0,0,1);
538         s1.add(s2);
539         assertEquals(s1.toString(), "0:01:000");
540     }
541
542     //Increase s2 to Min by adding s1
543     @Test
544     public void testAddStopWatchIncOneMinMethod () {
545         StopWatch s1 = new StopWatch(0,59,0);
546         StopWatch s2 = new StopWatch(0,1,0);
547         s2.add(s1);
548         assertEquals(s2.toString(), "1:00:000");
549     }
550
```

```
551     //Increase s2 to Min by adding s1
552     @Test
553     public void testAddStopWatchIncAddMinMethod () {
554             StopWatch z = new StopWatch(10,0,0);
555             StopWatch x = new StopWatch(20,0,0);
556             z.add(x);
557             assertEquals(z.toString(), "30:00:000");
558         }
559
560     //Increases s by 999 millisec
561
562     @Test
563     public void testAddMillIncMethod () {
564         StopWatch t  = new StopWatch();
565
566         for(int i = 0; i < 999; i++)
567         {
568             t.inc();
569
570         }
571         assertEquals(t.toString(), "0:00:999");
572     }
573
574     //Increases s by 1 sec
575     @Test
576     public void testAddSecIncMethod () {
577         StopWatch s1  = new StopWatch();
578
579         for(int i = 0; i < 1000; i++)
580         {
581             s1.inc();
582
583         }
584
585         assertEquals(s1.toString(), "0:01:000");
586
587     }
588
589
590     //Increases s1 to 1 Min
591     @Test
592     public void testAddMinIncMethod () {
593         StopWatch s1  = new StopWatch();
594         for(int i = 0; i < 60000; i++)
595         {
596             s1.inc();
597
598         }
599
600         assertEquals(s1.toString(), "1:00:000");
601     }
602
603     @Test
604     public void testAddFifyNineSecIncMethod () {
605         StopWatch s  = new StopWatch();
```

```
606         for(int i = 0; i < 59000; i++)
607         {
608             s.inc();
609 
610         }
611 
612         assertEquals(s.toString(), "0:59:000");
613     }
614 
615 
616     @Test
617     public void testEqualWithEqualCase () {
618         StopWatch s1 = new StopWatch (5,59,300);
619         StopWatch s4 = new StopWatch (5,59,300);
620 
621         assertTrue (s1.equals(s4));
622         assertTrue (s4.equals(s1));
623 
624     }
625 
626     //Test Objects with Different Mins but everything else being equal
627     @Test
628     public void testEqualTestMins() {
629     StopWatch s5 = new StopWatch (0,0,0);
630     StopWatch s6 = new StopWatch (1,0,0);
631     assertFalse(s5.equals(s6));
632     assertFalse(s6.equals(s5));
633     }
634 
635     //Test Objects with Different Sec everything else being equal
636     @Test
637     public void testEqualTestSec() {
638         StopWatch s7 = new StopWatch (5,47,300);
639         StopWatch s8 = new StopWatch (5,49,300);
640         assertFalse(s7.equals(s8));
641         assertFalse(s8.equals(s7));
642     }
643 
644     //Test Objects with Different Mill everything else being equal
645     @Test
646     public void testEqualTestMil() {
647         StopWatch s9 = new StopWatch (5,59,417);
648         StopWatch s10 = new StopWatch (5,59,624);
649         assertFalse(s9.equals(s10));
650         assertFalse(s10.equals(s9));
651     }
652 
653     @Test
654     public void testCompareToEqualCase () {
655         StopWatch s1 = new StopWatch (5,59,300);
656         StopWatch s4 = new StopWatch (5,59,300);
657 
658         assertTrue (s1.compareTo(s4) == 0);
659 
660     }
```

```
661
662     //Test compare different Min everything else being equal
663     @Test
664     public void testCompareToMin () {
665         StopWatch s11 = new StopWatch (6,0,0);
666         StopWatch s12 = new StopWatch (5,0,0);
667         assertTrue (s11.compareTo(s12) > 0);
668         assertTrue (s12.compareTo(s11) < 0);
669     }
670
671     //Test compare different Sec everything else being equal
672     @Test
673     public void testCompareToSec () {
674         StopWatch s11 = new StopWatch (0,59,999);
675         StopWatch s12 = new StopWatch (0,58,999);
676         assertTrue (s11.compareTo(s12) > 0);
677         assertTrue (s12.compareTo(s11) < 0);
678     }
679
680
681     //Test compare different Min everything else being equal
682     @Test
683     public void testObjectEqualMin () {
684         Object s13 = new StopWatch (6,0,0);
685         StopWatch s14 = new StopWatch (5,0,0);
686         assertFalse(s14.equals(s13));
687         assertFalse(s13.equals(s14));
688
689     }
690
691     //Test compare different Sec everything else being equal
692     @Test
693     public void testObjectEqualSec () {
694         Object s15 = new StopWatch (0,59,999);
695         StopWatch s16 = new StopWatch (0,58,999);
696         assertFalse(s16.equals(s15));
697         assertFalse(s15.equals(s16));
698     }
699
700     //Test compare different Mil everything else being equal
701     @Test
702     public void testObjectEqualMil () {
703         Object s17 = new StopWatch (0,59,999);
704         StopWatch s18 = new StopWatch (0,59,998);
705         assertFalse(s18.equals(s17));
706         assertFalse(s17.equals(s18));
707     }
708
709     //Test equal case
710     @Test
711     public void testObjectEqual () {
712         Object s19 = new StopWatch (0,59,999);
713         StopWatch s20 = new StopWatch (0,59,999);
714         assertTrue(s20.equals(s19));
715         assertTrue(s19.equals(s20));
```

```
716
717     }
718
719
720     @Test
721     public void testLoadSave () {
722         StopWatch s1 = new StopWatch (5,59,300);
723         StopWatch s2 = new StopWatch (5,59,300);
724
725         s1.save("file1");
726         s1 = new StopWatch ();  // resets to zero
727         s1.load("file1");
728         assertTrue (s1.equals(s2));
729
730     }
731
732     //Trys to add millisec while mutate is false
733     @Test
734     public void testMutateAddMil () {
735         StopWatch w = new StopWatch (5,59,300);
736         StopWatch v = new StopWatch (5,59,300);
737
738         StopWatch.setMutate(false);
739         w.add(1000);
740         assertTrue (w.equals(v));
741         StopWatch.setMutate(true);
742         }
743
744     //Trys to set min while mutate is false
745         @Test
746         public void testMutateSetMin () {
747             StopWatch w = new StopWatch (5,59,300);
748             StopWatch v = new StopWatch (5,59,300);
749
750             StopWatch.setMutate(false);
751             w.setMinutes(55);
752             assertTrue (w.equals(v));
753             StopWatch.setMutate(true);
754             }
755
756         //Trys to set sec while mutate is false
757         @Test
758         public void testMutateSetSec () {
759             StopWatch w = new StopWatch (5,59,300);
760             StopWatch v = new StopWatch (5,59,300);
761
762             StopWatch.setMutate(false);
763             w.setSeconds(40);
764             assertTrue (w.equals(v));
765             StopWatch.setMutate(true);
766             }
767
768         //Trys to set millisec while mutate is false
769         @Test
770         public void testMutateSetMil () {
```

```
771             StopWatch w = new StopWatch (5,59,300);
772             StopWatch v = new StopWatch (5,59,300);
773
774             StopWatch.setMutate(false);
775             w.setMilliseconds(500);
776             assertTrue (w.equals(v));
777             StopWatch.setMutate(true);
778         }
779
780         //Trys to add two StopWatch objects mutate is false
781         @Test
782         public void testMutateAddStopWatch () {
783             StopWatch w = new StopWatch (5,59,300);
784             StopWatch v = new StopWatch (5,59,300);
785
786             StopWatch.setMutate(false);
787             w.add(v);
788             assertTrue (w.equals(v));
789             StopWatch.setMutate(true);
790         }
791
792         //Trys to use Increment method mutate is false
793         @Test
794         public void testMutateInc () {
795             StopWatch w = new StopWatch (5,59,300);
796             StopWatch v = new StopWatch (5,59,300);
797
798             StopWatch.setMutate(false);
799             w.inc();
800             assertTrue (w.equals(v));
801             StopWatch.setMutate(true);
802         }
803 }
804
805
```

```java
package package1;
import javax.swing.JFrame;
import static org.junit.Assert.assertTrue;

//import static org.junit.Assert.assertEquals;

public class driver {

    public static void main(String[] args) {

        StopWatch s = new StopWatch ("20:10:8");
        System.out.println("Time: " + s);

        s = new StopWatch("20:8");
        System.out.println("Time: " + s);

        s = new StopWatch("8");

        System.out.println("Time: " + s);

        StopWatch s1 = new StopWatch(20, 2, 200);
        System.out.println("Time: " + s1);

        if(!StopWatch.equals(s, s1));
            System.out.println("s is Not equal to s1");

        s1.add(1000);
        System.out.println("Time: " + s1);

        StopWatch s2 = new StopWatch(40, 10, 20);
        s2.add(100);
        for (int i = 0; i < 4000; i++)
            s2.inc();
        System.out.println("Time: " + s2);

        s1.setMinutes(59);
        System.out.println("Time: " + s1.toString());

        s1.setSeconds(59);
        System.out.println("Time: " + s1.toString());

        s1.setMilliseconds(999);
        System.out.println("Time: " + s1.toString());

        StopWatch s3 = new StopWatch(10, 20);
        System.out.println("Time: " + s3.toString());

        StopWatch s4 = new StopWatch(20);
        System.out.println("Time: " + s4.toString());

        s4.add(60000);
        s4.add(1000);

        System.out.println("Time: " + s4.toString());

```

```
56          s4.add(s1);
57
58          System.out.println("Time: " + s4.toString());
59
60          for(int i = 0; i < 1000; i++)
61          {
62              s1.inc();
63
64          }
65
66          System.out.println("Time: " + s1.toString());
67
68          System.out.println(s4.equals(s3));
69
70          System.out.println(s1.compareTo(s4));
71
72          Object s17 = new StopWatch (0,59,999);
73          StopWatch s18 = new StopWatch (0,59,998);
74          System.out.println(s18.equals(s17));
75
76
77          StopWatch w = new StopWatch (5,59,300);
78          StopWatch v = new StopWatch (5,59,300);
79
80          StopWatch.setMutate(false);
81          w.setSeconds(40);
82          System.out.println(w.toString());
83          StopWatch.setMutate(true);
84
85          JFrame frame = new JFrame("Timer");
86          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
87
88          frame.getContentPane().add(new MyTimerPanel());
89
90          frame.pack();
91          frame.setVisible(true);
92      }
93
94  }
95
```

```java
package package1;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;


public class MyTimerPanel extends JPanel{
    private JButton start;
    private JButton stop;
    private JButton reset;
    private JLabel current;
    private StopWatch s1;
    private Timer javaTimer;




    public MyTimerPanel(){
        start =  new JButton("Start");
        stop = new JButton("Stop");
        reset = new JButton("Reset");
        current = new JLabel(" ");
        s1 = new StopWatch(0,0,0);
        javaTimer = new Timer(1, new TimerListener());

        ButtonListener listener = new ButtonListener();
        start.addActionListener(listener);
        stop.addActionListener(listener);
        reset.addActionListener(listener);



        add(start);
        add(stop);
        add(reset);
        add(current);




        setBackground(Color.cyan);
        setPreferredSize(new Dimension(500,240));
    }

    private class ButtonListener implements ActionListener
    {
        public void actionPerformed(ActionEvent event){
            if(event.getSource() == stop)
            {
                javaTimer.stop();
                current.setText(s1.toString());
            }

            if(event.getSource() == start)
```

```
56              {
57                  current.setText("");
58                  javaTimer.start();
59              }
60
61              if(event.getSource() == reset)
62              {
63                  s1 = new StopWatch(0,0,0);
64                  current.setText("");
65              }
66          }
67
68      }
69
70      private class TimerListener implements ActionListener
71      {
72          public void actionPerformed(ActionEvent e)
73          {
74              s1.inc();
75          }
76      }
77  }
78
```