CIS 163 Project 1
A Count Down Timer program

Purpose:

Grand Rapids city is in need of your help! Every year they have a new year's event downtown, but seem to have a problem with getting the timing down. That is, they need a program to help them know exactly when the new year begins and how much time they have left to get the city ready. Your task is to write a countdown timer class to help out the city of GR!

Due Date

- At the beginning of the lab; see the schedule, last page of the syllabus.

Before Starting the Project

- Review Chapter 1 – 6 of the CIS163 book
- Read this entire project description before starting

Learning Objectives

After completing this project you should be able to:

- *have a good working knowledge of the topics covered in CIS162*
- *create classes with associated methods*
- *use complex* if statements
- *read and write data* from external text files
- *use* static methods and properties available in the Java library
- *use the internet and API to create a Timer object*

You must complete each step fully before proceeding on. No credit is given to any given step unless the previous steps have been completed and the step is functioning correctly!

Before you turn in your work: follow the Java Style Guide to correctly document your project. (10 pts)

Step 1: Create an Eclipse project named "CountDownTimerPrj"

- Create a package named: project1 (right click on "CountDownTimerPrj" and select new/package)
- Create a class named: CountDownTimer (right click on "project1" and select new/class)
  - The properties and methods for this class are in step 2.
- Create a JUnit Test Case named: CountDownTimerTest (right click on "project1"and select new/JUnit Test Case)
  - Log on to BB and cut and paste the file found in the project 1 folder under Course Documents.

Step 2: Implement the following methods for the class "CountDownTimer":

Implement the following methods and properties in the CountDownTimer class. For properties, you will need three instance variables: hours (integer), minutes (integer), seconds (integer). For methods, you will need to implement the following (include any setters or getters that are needed). Unless otherwise stated, you can assume the input has no errors (i.e., a valid set of numbers) contained within.

- `public CountDownTimer()` Default constructor that sets the CountDownTimer to zero.
- `public CountDownTimer(int hours, int minutes, int seconds)` A constructor that initializes the instance variables with the provided values.
- `public CountDownTimer(int minutes, int seconds)` A constructor that initializes the instance variables with the provided values. Initialize hours to 0.
- `public CountDownTimer(int seconds)` A constructor that initializes the instance variables with the provided values. Initialize hours and minutes to 0.
- `public CountDownTimer (CountDownTimer other)` A constructor that initializes the instance variables with the other CountDownTimer parameter.
- `public CountDownTimer(String startTime)` A constructor that accepts a String as a parameter with the following format: "1:21:30" where 1 indicates hours, 21 indicates minutes, and 30 indicates seconds. OR the format "15:20" where the 15 indicates minutes, and 20 indicates seconds, OR the format "30" where 30 indicates seconds. If a value is not specified, i.e., "", then it is set to zero. You can assume the input has no errors (i.e., a valid set of numbers) contained within.
- `public boolean equals(Object other)` A method that returns true if "this" CountDownTimer object is exactly the same (mins = other.mins && secs = other.secs && …) as the other object (Note: you must cast the other object as a CountDownTimer object).
- `public static boolean equals(CountDownTimer t1, CountDownTimer t2)` A static method that returns true if CountDownTimer object t1 is exactly the same as CountDownTimer object t2 (i.e. mins = other.mins && secs = other.secs && …).
- `public int compareTo(CountDownTimer other)` A method that returns 1 if "this" CountDownTimer object is greater than the other CountDownTimer object; returns -1 if the "this" CountDownTimer object is less than the other CountDownTimer; returns 0 if the "this" CountDownTimer object is equal to the other CountDownTimer object.
- `public static int compareTo(CountDownTimer t1, CountDownTimer t2)` A method that returns 1 if CountDownTimer object t1 is greater than CountDownTimer object t2; returns -1 if the CountDownTimer object t1 is less than CountDownTimer t2; returns 0 if the CountDownTimer object t1 is equal to CountDownTimer object t2.
- `public void sub(int seconds)` A method that subtracts the number of seconds from "this" CountDownTimer object. You may assume the parameter "seconds" is positive.
- `public void sub (CountDownTimer other)` A method that subtracts CountDownTimer other from the "this" CountDownTimer object.
- `public void dec()` A method that decrements the "this" CountDownTimer by 1 second.
- `public void add(int seconds)` A method that adds the number of seconds to "this" CountDownTimer object. You may assume the parameter "seconds" is positive.
- `public void add (CountDownTimer other)` A method that adds CountDownTimer other to the "this" CountDownTimer object. Convert the "other" CounterDownTimer object to seconds and call the add (int seconds) method. You may assume the parameter "seconds" is positive.
- `public void inc()` A method that increments the "this" CountDownTimer by 1 second.
- `public String toString()` A method that returns a string that represents the state of a CountDownTimer with the following format: "1:06:01". Display the hours as is; minutes with 2 digits including a leading "0" if minutes < 10, and seconds with 2 digits again including a leading "0" if seconds < 10. Other examples: "21:32:00", "0:00:00".

Step 3: Software Testing: Using a JUnit named "TestCountDownTimer" Software developers must plan from the beginning that their solution is correct.

- Within this file you will see comments on where to place the JUnit test cases.

Step 4: Software Testing: Using a main program

A start to a main program has been provided (see below) and your task is to add on **many** Java statements that would test each method separately and completely.

Main Method

```
public static void main(String[] args) {
        CountDownTimer s = new CountDownTimer("2:59:8");
        System.out.println("Time: " + s);

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

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

        CountDownTimer s1 = new CountDownTimer(25, 2, 20);
        System.out.println("Time: " + s1);

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

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

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

}               // ADD more test cases here
```

Sample Results

```
Time: 2:59:08
Time: 0:20:08
Time: 0:00:08
Time: 25:02:20
Time: 24:45:40
Time: 25:02:20
Time: 39:02:00
```

Now, compare Step 3's approach to testing to Step 4's approach. Is one better than the other? Also, remember to use the Java Style Guide for documenting your program.

****

Step 5: Create the following additional methods in the CountDownTimer class:

- `public void save(String fileName)` A method that saves the "this" CountDownTimer to a file; use the parameter filename for the name of the file.
- `public void load(String fileName)` A method that loads the "this" CountDownTimer object from a file; use the parameter filename for the name of the file.
- `public static void Suspend(Boolean flag)` A method that turns 'off' and 'on' any add method in CountDownTimer. In other words, when the flag is true then it prevents any add method from changing (mutate) the state of the "this" object as it relates to hours, minutes, and seconds. If flag is false, then mutation is allowed.

- **A change from step 2**; allow for an error in the input arguments for **all** constructors and methods, and throw a IllegalArgumentException exception if an error occurs. For example, "12:123:30" is not a valid input string for a constructor in step 2 and an exception is thrown. Another example, if you subtract below Zero, throw an error.
    - Sample code snippet many help:

```
if (min >= 60)
        throw new IllegalArgumentException();
```

Here is some pseudo code that would read/write to a file:

```
public void sampleloadData(String fileName){
	int someInt;

	try{
		// open the data file
		Scanner fileReader = new Scanner(new File(fileName));
		Scanner lineReader;

		// read one int
		someInt = fileReader.nextInt();
		System.out.println (someInt);
		someInt = fileReader.nextInt();
		System.out.println (someInt);
	}

	// problem reading the file
	catch(Exception error){
		throw new IllegalArgumentException();
	}
}
```

Here is the code that would write the above file:

```
public void sampleSaveData (String fileName) {
	PrintWriter out = null;
	try {
		out = new PrintWriter(new BufferedWriter(new FileWriter(fileName)));
	}
	catch (Exception e) {
		throw new IllegalArgumentException();
	}
	out.println(42);
	out.println(163);
	out.close();
	}
```

Step 6: Software Testing: Complete the second part of the JUnit class named: TestCountDownTimer.

- Again, read the comments contained in the CountDownTimerTest file. BE SURE to include JUnits test cases that show your CountDownTimer is properly throwing exceptions from step 5. Note: one JUnit test per exception!

Step 7: Challenge Requirement, read chapter 6 in your book.

- The following should only be attempted after all of the other requirements (previous steps) have been completed.
- Create a GUI front end to your project and create 3 CountDownTimers with associated JButtons so that each CountDownTimer can be started, stopped, and have new values entered for the start time. **Important**, have buttons on the GUI that show that all the methods in Step 2 are working correctly. For example, have a button that increments the CountDownTimers by 1, this button uses the inc() method in step 2.
- The following is a suggestion on how to work with the CountDownTimer class. This is only a suggestion.

```
public class MyTimerPanel extends JPanel {
   private CountDownTimer countdowntimer;
   private Timer javaTimer;
   private TimerListener timer;
   …..

   public MyTimerPanel() {
      countdowntimer = new CountDownTimer(0, 0, 10);
      timer = new TimerListener();
      javaTimer = new Timer(1000, timer);
      javaTimer.start();
         ….
   }

   private class TimerListener implements ActionListener {
      public void actionPerformed(ActionEvent e) {
         …..
```

- **Suggestion**: Create one JPanel (named: CDTpanel) with one CountDownTimer, in this panel. Create all the needed buttons to control that one CounterDownTimer object (make this one panel look good). Then create a new class call (MyTimerPanel) with 3 JPanels, and instantiate CDTpanel 3 times. Please ask your professor if you want to see a demo of this tip.

Research how the Swing.Timer class works using google and see your instructor for more details.

- Recommend website: http://www.java2s.com/Code/Java/Swing-JFC/TimerSample.htm)
- Create an actual CountDownTimer that displays time every 1 second.

---------- YOUR'RE DONE ☺ ----------

Some additional grading criteria

There is a 70% penalty on programming projects if your solution does not compile.

Late Policy

Projects are due at the START of the class period and the first 24 hours (-20 pts)

- Each subsequent weekday is an additional -10 pts

Project 1: "CountDownTimer" Program Rubric.

Student Name	
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	
Steps 1 – 2: Basic Functionality • public CountDownTimer() • public CountDownTimer(int hours,int minutes,int seconds) • public CountDownTimer(int minutes, int seconds) • public CountDownTimer(int seconds) • public void CountDownTimer (CountDownTimer other) • public CountDownTimer(String startTime) • public boolean equals(Object other) • public int compareTo(CountDownTimer other) • public subtract operations • public add operations. • public String toString()	40	
Steps 3,6: JUnit test	15	
Step 4: Main test	5	
Step 5: Added functionality • public void save(fileName), • public void load(fileName) • public static void Suspend() • Check for an errors in the input for all constructors and methods	4 6 3 7	
Step 7: Challenge Requirement	10	
Total	**100**	

Additional Comments: