**CIS 163 Project 1**

**A StopWatch program**

**Due Date**

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

**Before Starting the Project**

* Review Chapter 1 – 5 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 are functioning!**

**Before you turn in your work: use the Java Style Guide to document your project.**

**Step 1: Create an Eclipse project named “StopWatchPrj”**

* Create a package named: project1 (right click on “StopWatchPrj” and select new/package)
* Create a class named: StopWatch (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: TestStopWatch (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 “StopWatch”:**

Implement the following methods and properties in StopWatch class:

For properties, you will need at least these three instance variables:

* int minutes
* int seconds
* int milliseconds

For methods, you will need to implement the following (include getters and setters as needed). At this point, assume all values of parameters are valid (i.e., within range) numbers.

* public StopWatch() Default constructor that sets the StopWatch to zero
* public StopWatch(int minutes, int seconds, int milliseconds) A constructor that initializes the instance variables with the provided values
* public StopWatch(int seconds, int milliseconds) A constructor that initializes the instance variables with the provided values
* public StopWatch(int milliseconds) A constructor that initializes the instance variables with the provided values
* public StopWatch(String startTime) A constructor that accepts a string as a parameter with the following format: “1:21:300” where 1 indicates minutes, 21 indicates seconds, and 300 indicates milliseconds. OR the format “15:200” where the 15 indicates seconds, and 200 indicates milliseconds, OR the format “300” where 300 indicates milliseconds. If a value is not specified, then it is set to zero. You can assume the input has no errors (i.e., a valid set of numbers) contained with in.
* public boolean equals (StopWatch other) A method that returns true if “this” StopWatch object is the same as the other StopWatch object (i.e., this.minutes equals other.minutes and this.seconds equals other.seconds and so on).
* public boolean equals (Object other) A method that returns true if “this” StopWatch object is the same as the other object;

this.minutes equals ((StopWatch) other).minutes and so on.

* public static boolean equals (StopWatch s1, StopWatch s2) A static method that returns true if StopWatch object s1 is exactly the same as StopWatch object s2.
* public int compareTo (StopWatch other) A method that returns 1 the time represented by “this” StopWatch object is greater than the time represented by the other StopWatch; returns -1 if the time represented by “this” StopWatch object is less than the other StopWatch; returns 0 if the time represented by “this” StopWatch object is equal to the other StopWatch object
* public void add (int milliseconds) A method that adds the number of milliseconds to “this” StopWatch object. You may assume in this step the parameter “milliseconds” is positive. Suggestion: call the inc method within a loop.
* public void add (StopWatch other) A method that adds StopWatch other to the “this” StopWatch. Suggestion: convert “other” to milliseconds and call the add(int milliseconds) method
* public void inc ()A method that increments the “this” StopWatch by 1 millisecond
* public void sub (int milliseconds) A method that subtracts the number of milliseconds from “this” StopWatch object. You may assume in this step the parameter “milliseconds” is positive. Suggestion: call the dec method within a loop.
* public void sub (StopWatch other) A method that subtracts StopWatch other from the “this” StopWatch. Suggestion: convert “other” to milliseconds and call the sub(int milliseconds) method
* public void dec () A method that decrements the “this” StopWatch by 1 millisecond
* public String toString () Method that returns a string that represents a StopWatch with the following format: “1:06:010”. Display the minutes as is; if seconds < 10 then display with a leading “0”, and always display milliseconds with 3 digits. Other examples: “21:32:000”, “0:00:000”

**Step 3: Software Testing: Using a JUnit named “TestStopWatch”:**

Software developers must plan from the beginning that their solution is correct. Assume on this step, the user is entering valid input, e.g., add(10) or sub (20). Assume seconds is always between 0 and 59, milliseconds is always between 0 and 999 for constructors. However, add(milliseconds) and/or sub(milliseconds) is the assumption is greater 0 for milliseconds.

* Add on JUnit test cases that will test the functionality of the methods created in step 2.

**Step 4: Create the following additional methods in the StopWatch class:**

* public void save(String fileName) A method that saves the “this” StopWatch to a file
* public void load(String fileName)A method that loads the “this” StopWatch from a file
* public static void suspend(boolean flag)This method suspends all operations (add, sub, inc, dec) when the flag = true. Otherwise all StopWatch objects can be mutated. (Hint: you will need to add a static variable, make it private)

Following will help you with reading and writing to a file:

The data file is shown directly below and contains only one line of data. Listing 6.30 of your book shows the use of the Scanner Class.

Here is the code that would read from the specified 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);

}

// problem reading the file

**catch**(Exception error){

System.*out*.println("Oops! Something went wrong.");

}

}

Here is the code that would write to the specified file.

**public** **void** sampleSaveData (String fileName) {

PrintWriter out = **null**;

**try** {

out = **new** PrintWriter(**new** BufferedWriter(**new** FileWriter(fileName)));

}

**catch** (Exception e) {

e.printStackTrace();

}

out.println(163);

out.close();

}

**Step 5: Complete error checking for Step 2 (StopWatch class):**

Check for errors in the input and throw an IllegalArgumentException() for all of the methods in step 2. Assume on this step, the user is entering invalid input. Throw an exception if the seconds is not between 0 and 59, and/or milliseconds is not between 0 and 999 for constructors. Throw an exception if milliseconds is less than zero for add(milliseconds) and/or sub(milliseconds).

* + Examples: sub(-10), add(-10), etc.

**Step 6: Software Testing: Complete the second part of the JUnit class named, TestStopWatch:**

* Add on JUnit test cases that will test the functionality of the methods created in steps 4 and the extra error checking in step 5.

**Step 7: Challenge Requirement:**

* **The following must only be attempted after all of the other requirements have been completed.**
* Create a GUI front end to your projects so you can start, stop, add, sub, load (use JFileChooser), save (use JFileChooser), inc, dec and reset your stopwatch.
* Have at least three StopWatch objects on the panel.
* One JButton to suspend all stopwatches.

Implement this GUI only after all other requirements have been completed. Please review Chapters 6 in your book

**JPanel for one StopWatch**

* Create a JPanel to load, save, start, stop, sub, add, inc, dec to a single StopWatch. Include a private inner class that implements ActionListener to handle button clicks (see Ch 6.1 – 6.3). The instructor will demonstrate this technique in class. See MyTimerPanel below.
* (optional – gold star) Within actionPerformed(), catch two exceptions: NumberFormatException and IllegalArgumentException. This will prevent common errors of not providing valid integers in the text fields or using illegal arguments to add() or sub(). Display appropriate JOptionPane error messages. Here too, the instructor will demonstrate this in class.

public class MyTimerPanel extends JPanel {

private StopWatch stopWatchTimer;

private Timer javaTimer;

private TimerListener timer;

…..

public MyTimerPanel() {

stopWatchTimer = new StopWatch (0, 0, 0);

timer = new TimerListener();

javaTimer = new Timer(1000, timer);

javaTimer.start();

….

}

private class TimerListener implements ActionListener {

public void actionPerformed(ActionEvent e) {

…..

Final step, create a new JPanel that will create 3 MyTimerPanels. Then from the main, call this new MyTimerPanel. Here too, the instructor will demonstrate this in class.

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

* I recommend: http://www.java2s.com/Code/Java/Swing-JFC/TimerSample.htm)

--------------------------- YOUR’RE DONE ☺ -------------------------------

**Project 1: “StopWatch” 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/java-coding-style-guide/>)   * 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** | 40 |  |
| **Step 3: JUnit test** | 8 |  |
| **Step 4: Added functionality**   * public void save(String fileName) * public void load(String fileName) * public static void suspend(boolean flag) | 12 |  |
| **Step 5: Error checking everything** | 10 |  |
| **Step 6: More Software Testing (JUnit)** | 10 |  |
| **Step 7: Challenge Requirement** | 10 |  |
| **Total** | **100** |  |

**Additional Comments:**