**CIS 163 Project 1**

**A Geo Count Down Timer program**

**Project description:**

As a developer you are tasked to create an API that will allow users to create a count down timer for dates that are very far in the future. For example, this year we had a very cold winter. Some people would say “it was a 100-year winter”, meaning: the next winter like this one will be in the year 2119. Your task is to create a “GeoCountDownTimer” class that allows users to set a date in the future; i.e., GeoCountDownTimer(“10/30/2116”). As with any API that will be used by millions of applications, you will need to thoroughly test the available operations. Developers can be very unpredictable in how they use your API so the more creative you can be in your testing scenarios the stronger your API will be. This API is very similar to existing Open Source libraries with an interesting specific twist. This will provide you with insights as to how these libraries are developed and tested.

**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*

**Before you turn in your work: use the** [**Java Style Guide**](http://www.cis.gvsu.edu/studentsupport/javaguide) **to document your project. (10 pts)**

**Step 1: Create an IntelliJ project named “GeoCountDownTimerPrj”**

* Create a package named: project1 (right click on “GeoCountDownTimerPrj” and select new/package)
* Create a class named: GeoCountDownTimer (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: TestGeoCountDownTimer (This is tricky, please attend lab and the instructor will demonstrate how best to do this)
  + 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 “GeoCountDownTimer”**

Implement the following methods and properties in GeoCountDownTimer class. For properties, you will need three instance variables: years (integer), months (integer), and days (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. (This assumption is only true for step 2; see step 5 for a change)

* private GeoCountDownTimer() Default constructor that sets the GeoCountDownTimer to zero. *(This is interesting, a private constructor, do you know why this would be useful?)*
* public GeoCountDownTimer(int month, int day, int year) A constructor that initializes the instance variables with the provided values.
* public void GeoCountDownTimer (GeoCountDownTimer other) A constructor that initializes the instance variables with the other GeoCountDownTimer parameter.
* public GeoCountDownTimer(String geoDate) A constructor that accepts a string as a parameter with the following format: “5/10/2019” where 5 indicates the month, 10 indicates the day, and 2019 indicates the year. 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” GeoCountDownTimer object is exactly the same as the other object (Note: you must cast the other object as a GeoCountDownTimer object).
* public int compareTo(GeoCountDownTimer other) A method that returns 1 if “this” GeoCountDownTimer object is greater than the other GeoCountDownTimer object; returns -1 if the “this” GeoCountDownTimer object is less than the other GeoCountDownTimer; returns 0 if the “this” GeoCountDownTimer object is equal to the other GeoCountDownTimer object. For example: “1/20/2010” is less than 12/31/2011.
* public void dec(int days) A method that subtracts the number of days from the “this” GeoCountDownTimer object This will adjust the timer date down the number of days. (E.g. Current date 1/10/2015 will be 1/5/2015 if passing in 5)
* public void dec ()A method that subtracts 1 day from the “this” GeoCountDownTimer object.
* public void inc(int days) A method that adds the number of days to the “this” GeoCountDownTimer object This will adjust the timer date up the number of days. (E.g. Current date 1/10/2015 will be 1/15/2015 if passing in 5).
* public void inc ()A method that adds 1 day from the “this” GeoCountDownTimer object.
* public String toString() Method that returns a string that represents a GeoCountDownTimer with the following format: “month day, year”. For example: February 10, 2019
* public String toDateString() Method that returns a string that represents a GeoCountDownTimer with the following format: “month/day/year”. For example: 2/20/2019.

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

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. To receive full credit, you must have at least 2 pages of testing methods.

**Main Method**

GeoCountDownTimer s = **new** GeoCountDownTimer(**"2/10/2018"**);  
 System.***out***.println(**"Date: "** + s);  
  
 GeoCountDownTimer s1 = **new** GeoCountDownTimer(2, 20, 2019);  
 System.***out***.println(**"Date: "** + s1.toDateString());  
  
 s1.inc(365);  
 System.***out***.println(**"Date: "** + s1);  
  
 GeoCountDownTimer s2 = **new** GeoCountDownTimer(4, 10, 2019);  
 **for** (**int** i = 0; i < (366 + 365+ 365 + 365); i++)  
 s2.inc();  
 System.***out***.println(**"Date: "** + s2);  
  
 *// Create many more test cases in this driver method to  
 // prove the class is functioning correctly.*}

**Sample Results**  
Date: February 10, 2018

Date: 2/20/2019

Date: February 20, 2020

Date: April 10, 2023

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**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 GeoCountDownTimer class:**

* public void save(String fileName)A method that saves the “this” GeoCountDownTimer to a file; use the parameter filename for the name of the file.
* public void load(String fileName)A method that loads the “this” GeoCountDownTimer object from a file; use the parameter filename for the name of the file.
* public int daysToGo(String fromDate) A method that returns the number of days to go from the parameter “fromDate” (typically today’s date, but not always) to the “this” object. For example:

GeoCountDownTimer s1 = **new** GeoCountDownTimer (2,9,2019);

*System.out.println* (s1.daysToGo("2/1/2019"));

*Output: 8*

*// There are 8 days from the parameter date*

*// of 2/1/2019 to the “this” date of 2/9/2019.*

Finally, if the fromDate is greater than the “*this”* date, throw IllegalArgumentException.

* public GeoCountDownTimer daysInFuture(int n) A method that returns a GeoCountDownTimer given is the number of days in the future (if n < 0 =, then return a GeoCountDownTimers given the number of days in the past). For example:

GeoCountDownTimer s1 = **new** GeoCountDownTimer (12,9,2019);

*System.out.println* (s1. daysInFuture (10).toDateString());

*Output: 12/19/2019*

Finally, if the fromDate is greater than the “*this”* date, throw IllegalArgumentException.

* **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. **The year cannot be less than 2019**.
  + For example:

if (days <= 0)

throw new IllegalArgumentException();

**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.

ANCHORAGE 256000

**Here is the code that would read the above file:**

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

String city;

**int** population;

**try**{

// open the data file

Scanner fileReader = **new** Scanner(**new** File(fileName));

// read one String in of data and an int

city = fileReader.next();

population = fileReader.nextInt();

System.*out*.println (city + " " + population);

}

// could not find file

**catch**(Exception error) {

System.*out*.println("File not found ");

}

}

**Here is the code that would write the above file:**

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

PrintWriter out = **null**;

**try** {

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

}

**catch** (IOException e) {

e.printStackTrace();

}

String s = "ANCHORAGE";

out.println(s + " " + "256000");

out.close();

}

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

* Create a JUnit class that test all possible errors and valid inputs (see instructor regarding this requirement).

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

**YOU MUST SEE THE instructor regarding this step.**

* The following should only be attempted after all of the other requirements have been completed.
* Create a GUI front end to your project so you can start, stop and reset your GeoCountDownTimer.
  + You will need to speed up time to make this GUI functional.
* Try to create at least 3 random timers with a GUI front end.
* Create save and load buttons use JFileChooser.
* Create a inc and dec buttons.
* Display the number of days to go from a given date.

Create a class called “MyTimerPanel” that has a private inner class that implements ActionListener, see chapter 6 in your book; here is some help.

public class MyTimerPanel extends JPanel {

**private** GeoCountDown geoCountDownTimer;

**private** Timer javaTimer;

**private** TimerListener timer;

**public** MyTimerPanel() {

geoCountDownTimer = **new** GeoCountDown(5,10,2019);

timer = **new** TimerListener();

// This calls timer object 10 times per second

javaTimer = **new** Timer(10, timer);

// There is a problem here, your demonstration of your program could take years. So, you can

// assume that every time the timer object is called equals one day. So, in about 3 seconds would be

// a month. That is how you can speed up time.

javaTimer.start();

}

**private** **class** TimerListener **implements** ActionListener {

**public** **void** actionPerformed(ActionEvent e) {

}

}

}

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)
* Create an actual GeoCountDownTimer 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.

* Stapled cover page with your name and signed pledge. (-5 pts if missing)

**Late Policy**

Projects are due at the START of the class period and the first 24 hours (-15 pts)

* Each subsequent weekday is an additional -10 pts

**Turn In**

A professional document is stapled with an attractive cover page.

* Cover page - Your project must have a cover page that includes your name, a title, an interesting graphic or photograph related to the project topic and the following signed pledge: "I pledge that this work is entirely mine, and mine alone (except for any code provided by my instructor). " You are responsible for understanding and adhering to the [School of CIS Guidelines for Academic Honesty](http://www.cis.gvsu.edu/Academics/Honesty/).

**Project 1: “GeoCountDownTimer” 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 GeoCountDownTimer() * public GeoCountDownTimer(int month, int day, int year) * public void GeoCountDownTimer (GeoCountDownTimer other) * public GeoCountDownTimer(String geoDate) * public boolean equals(Object other) * public int compareTo(GeoCountDownTimer other) * public void dec(int days) * public void dec () * public void inc(int days) * public void inc () * public String toString() * public String toDateString() | 35 |  |
| **Step 3,6: JUnit test** | 10 |  |
| **Step 4: Main test** | 5 |  |
| **Step 5: Added functionality**   * public void save(String fileName) * public void load(String fileName) * public int daysToGo() * public GeoCountDownTimer daysInFuture(int n) * Allow for an error in the input for all constructors and methods   + **(approx. -2 pt per error found)** | 1  4  5  5  5 |  |
| **Step 7: Challenge Requirement (GUI)** | 10 |  |
| **UAT** | 10 |  |
| **Total** | **100** |  |

**Additional Comments:**