**Viewing assignment...**

▼ Settings for "Lab 02 - ExtClock"

| | |
|---|---|
| **Created by** | Dean Mellas |
| **Date created** | Jan 8, 2014 7:48 pm |
| **Open** | Jan 22, 2014 8:00 pm |
| **Due** | Jan 29, 2014 4:00 pm |
| **Accept Until** | Jan 29, 2014 4:05 pm |
| **Modified by instructor** | Jan 22, 2014 8:44 pm |
| **Student Submissions** | Attachments only |
| **Number of resubmissions allowed** | Unlimited |
| **Accept Resubmission Until** | Jan 29, 2014 4:05 pm |
| **Grade** | Points (max 10.0) |
| **Add due date to Schedule** | Yes |
| **Alert:** | No |
| **Honor pledge:** | No |

**Assignment Instructions**

Please note that my lab numbers are not based on the ordinal sequence of the assignments, but on the latest chapters upon which they are based. This first lab is named "Lab 02" because it requires reading through chapter 02 in the textbook (specifically, review Java Fundamentals) to complete. As this has now been done, the lab is officially assigned.

After completing the reading (and projects) in Chapters 1 and 2 of the textbook, design your own subclass of their Clock class as in Programming Exercise 1 (page 165 of the textbook, as amended, below).

Follow all their directions but only attach the Java source code file for the ExtClock class that you wrote (the class source code file that you attach must be named correctly as ExtClock.java). Be sure to include the following features (including my modifications, below):

The class must be named **ExtClock** and be derived from the Clock class already provided in the book (don't modify the author's Clock class). The source code is available on the publisher's web site. I have also included it as an attachment to this assignment. Also, all of the source code for the textbook is included in the Resources area of this TalonNet portal.

Be consistent: Your ExtClock must contain all of the features of, and comparable functionality to, the original Clock class, somehow, either as inherited, overridden, overloaded or new methods, but only as necessary to include the new Time Zone feature, properly, as the added functionality in the subclass. For example, the ExtClock non-default constructor and setTime method must now take four integers rather than only the original three, the last one being that of the Time Zone, but don't do any more work than is absolutely necessary in the subclass to implement the new feature(s) correctly (i.e., don't re-implement a feature if it isn't changing or doesn't need to be modified). Name the new field **zone** and implement it as an integer. Name the new accessors and mutators accordingly to our generally accepted Java naming and style standards (camelCase, full identifiers) - which are not necessarily those of the textbook author (i.e., don't use abbreviations for new field names, as the author does, and get- and set- method name endings should match those of the fields they modify).

Please be aware that the author SHOULD HAVE named his fields: **hour**, **minute** and **second** (by generally accepted Java standards). I pointed this out at the lecture. But even though we won't change the author's code, we will always follow those standards when we write our own (to maintain the programming convention and style habits the industry expects of us). He should have likewise named the get- and set- methods for the hour field as: **getHour** and **setHour** and the same goes for all the other get- and set- methods (note it's singular, not plural) but, what ever. Just don't forget.

Valid Time Zones are integers ranging from **-12** to **+12** (including **0**, so we will just ignore the zones that are not integers or that are greater than +12 even though some unusual zones with those features do, in fact, exist for reasons of commerce or politics). Be sure to add a method called **getZone**() to return the Time Zone as an integer. Also provide a **setZone**() method, even though the author didn't implement comparable mutator functionality in the original Clock class (you don't have to provide any set methods for the superclass fields that the author should have provided, but you should be able to figure out how this could have been accomplished in your subclass, if you did). Note that set- methods don't do any calculations, they just set the values specified.

Be sure method **toString** in ExtClock adds the Time Zone, properly. For example, if the time here in California were 7:30 PM, your toString() should return the String: **19:30:00 UTC-8** because the Time Zone for California is -8 (UTC-8). Reuse any code from Clock that you can (hint: using the super reference).

Don't forget to comment all your work, including Top-Comments for the file name, your name and a description of the new class. Comment how you are calculating the Time Zone adjustments (see below) in your source code.

MODIFICATIONS TO THE TEXTBOOK PROBLEM TO COMPLETE THE LAB:

Add a method **changeZone(int)** that takes a new Time Zone as an integer to set, **adjusting the current time** in the process, accordingly, to keep it consistent. For example, if the current Pacific or Los Angeles time is set as 19, 30, 0, -8 (that is, 19:30:00 UTC-8) and you change the time zone to -6 (Central or Chicago time) then the current time should adjust itself to 21:30:00 UTC-6 because Chicago is two hours ahead of us. Notice that if you change the Time Zone to +1 (Paris) your toString() should return the String: **04:30:00 UTC+1** because Paris is (usually) nine hours ahead of us (since we do not track any date, we do not need to account for daylight saving or summer time). Also notice that the plus sign shows for positive valued UTC Zones. The hours, minutes and seconds are zero padded, but the time zone is not, and, there is one space between the seconds and UTC, and no space between UTC and the plus or minus sign and the zone.

Make sure your ExtClock can adjust the Time Zone eastward (i.e., New York) and westward (i.e., Tokyo). Attach ExtClock.java only. I will compile and test with the author's Clock.java source code and my own test class (so make sure all the classes and methods are named properly - those are requirements specifications). Points off for everything that doesn't work, is named incorrectly or doesn't follow expected conventions of style (we will follow the ANSI for Allman indenting style with 3 spaces, never tabs, for each successive level of indenting - so, be sure to set your programmer's editor to substitute spaces for tabs).

Please note, this assignment tests very basic Java Fundamentals, coding style and naming convention skills. We haven't started any Data Structures, yet..

Good luck,
Dean.

**Additional resources for assignment**

Clock.java ( 5 KB; Jan 8, 2014 7:48 pm )

▸ Student view of the assignment "Lab 02 - ExtClock"