CS1110, Spring 2008, Assignment A4. Due Monday, March 8, by midnight on the CMS

Purpose

To give you practice with writing functions whose bodies have assignment statements, conditional statements, and blocks.

Ground rules

You may work with one partner. If you do, get on the CMS several days before the assignment is due and group yourselves. Do not wait until the last minute to do this. It is dishonest for partners to work independently on different parts of this assignment. You must work together on the whole assignment.

This is (roughly) a 10-hour assignment. Plan accordingly. Please keep track of your time and, just before you submit, tell us in a comment at the top of the class how many hours you spent. Spend time reading this handout so that you thoroughly understand what we are asking for before programming! There are important notes at the end of this document.

Academic integrity

You may talk in general terms to other people and discuss ideas about the assignment. But you should not look at or be in posession of someone else's program —whether that person is in the class or had a similar assignment in the past. Such dishonest behavior hurts you, since you are not learning and are hurting your own integrity, and us, because we have to grade something unnecessarily. For these reasons, we will not tolerate violations of academic integrity on this or any other assignment.

Class `Time`

You will write several functions in class `Time`. We give you a skeleton for the class, with all methods fully specified but with some stubs for bodies (empty body or simply a return statement), so that the program is syntactically correct and will compile. Look at them carefully; they are so complete that one can write the method body based on the specification. This is how you should write your method specifications. You can also see the Javadoc specs for the class. All of these are in this zip file but can also be dowloaded from the course website.

An instance of class Time represents a time in some time zone. The time zones that are implemented are:

- GMT: Greenwich Mean Time, GMT
- BST: British Summer Time, GMT+1
- EST: Eastern Standard Time, GMT-5 hours (NY)
- EDT: Eastern Daylight Savings Time, GMT-4 hours (NY)
- CST: Central Standard Time, GMT-6 hours (Chicago)
- CDT: Central Daylight Savings Time, GMT-5 hours (Chicago)
- MST: Mountain Standard Time, GMT-7 hours (Phoenix)
- MDT: Mountain Daylight Savings Time, GMT-6 (Phoenix)
- PST: Pacific Standard Time, GMT-8 hours (LA)
- PDT: Pacific Daylight Saving Time, GMT-7 hours (LA)
- IND: India time, GMT+5:30 hours (New Delhi) (Included to show that not all zones are on hourly boundaries)

To find out more about time zones, visit these web sites: abbreviations, world clock, daylight saving time, US law on time, time-zone map.

Some times may appear negative or greater than 24 hours. This is because of conversions from one time zone to another. For example, a time of 0 hours GMT is –7 hours PDT, while a time of 23:59 GMT is 29:29 IND. See the comment on the class for a complete specification.

An instance of the class can show the time using a 24-hour clock or using the AM-PM designation; it is the user's choice.

Study the specification of the class and its methods carefully in order to get an overall view of what the class is for. For this purpose, use either the skeleton or, better yet, the JAVADOC spec for it.

Create a JUnit testing class named TimeTester and use it to test all your methods. You will submit it on the CMS, and your grade will depend partly on the test cases that you use to test your methods. Please be thorough in your testing.

But we also provide a GUI, given by class TimeZoneGUI, that you can use to play around with your program, once it is finished. Execute

new TimeZoneGUI();

in the interactions pane to cause the GUI to appear on your monitor. Try it out. Later in the course, we will learn a bit about building such GUIs.

Download the files for this assignment from the course website or here.

This assignment will be presented as a series of tasks. **Finish each task completely before proceeding to the next**. Wherever it is possible, write each method in an incremental fashion. This holds especially for method toString, which is fairly complicated. Test each method carefully and thoroughly, using the JUnit testing class, before proceeding to the next.

Task 0: function objectName
This function should return the name on the tab of the object. This is used in the GUI to identify the objects.

Task 1: getter methods
You can see in the skeleton that we have given you two completed two constructors (and one that is uncompleted). Therefore, you can create instances of the class, but you can't get anything out of them. Your first task is to complete getter methods getTime, getZone, and getDisplayMode. Once they are properly written, check them out! This task should not be difficult.

Task 2: function toString.
Write function toString. You MUST follow its specification carefully. Don't just blindly make it do what you want; implement it to be consistent with the specification. To help you, we have put in the body of toString a sequence of statement-comments that, together, do what the body is supposed to do. All you have to do is implement these statement-comments. Put the implementation of each statement-comment AFTER the statement-comment. Of course, you may start with an empty body and do it the way YOU want. But it must be correct!

After your write function toString, test it thoroughly, using class TimeTester, with instances of the class for negative times, times in the range 0..24, and times greater than 24 hours and with hours, minutes, and seconds being 1 or 2 digits long. Also, test these with both modes of output —24-hour clock and 12-hour clock. This function MUST be correct before you proceed.

Task 3: function isLegal(String).
Users may give time zones that are not legal. The purpose of private function isLegal is to test whether its parameter is one of the legal time zones, returning true if it is and false if it is not. Implement isLegal. Test it on ALL legal zones, as well as illegal ones, before proceeding to the next step. Notice that the second constructor calls isLegal, so, you can test isLegal using this constructor. Hint: Here's one way to think about writing isLegal. Can you use one of the String functions to search for a zone in some String value?

Task 4: function isLegal().
If a time t is translated into a time in another zone, it may not appear legal in that new zone. The definition of legality is given in the specfication of this method.

Task 5: procedure setDisplay.
This procedure should be easy to write and test. It is a simple setter method.

Task 6: the third constructor.
Up to now, any instance you created had either time 0 or a time for which you gave a number of seconds. Now implement the third constructor, the one that has 5 parameters. It can be done with one statement. Test it thoroughly before proceeding!

Task 7: function timeInZone.
This function is given a time in one zone and is asked to create a new Time object that has the same time but in a different time zone. To help you out, take a look at function timeInGMT. It does roughly the same thing but always converts to GMT time. Function timeInGMT should give you an idea how to write function timeInZone. Here's a point to ponder. You have to translate from ANY time zone to ANY OTHER time zone. How many possibilities are there? Can you make use of function timeInGMT to simplify the task? Make sure you test this method thoroughly before proceeding in to the next task.

Task 8: function compareTo.
Implement function compareTo. Of course, it has to satisfy its specification. Note that time 18:00 GMT is the same as time 13:00 EST. You can't really compare the times until they are both converted to the same time zone. What's the simplest way to do that?

Task 9. Finish up.
You have written all the methods. You see how a class filled with methods can be written one method at a time, in order to come up with a neat little class. Now, go over your program once more. Make sure the indentation is correct — use DrJava's indenting feature if you want— and make sure that the comments are suitable (we gave most of them to you).

Finally, place a comment at the top of the class that tells us how many hours you worked on this assignment.

What to submit

Submit files `Time.java` and `TimeTester.java`. Before you do that, please place a comment at the top of the class that tells us roughly how many hours you worked on this assignment.

Notes

1. Restriction on times. Any time that is given in a new expression will be greater than –24 hours (– 1 day) and less than 48 hours (2 days) ---or else 0 is used instead. This restriction applies only to values given as arguments in constructor calls. That is the only place restrictions are given in the specification. An internal time may get out of that range. For example, if you create a folder with time 47 hours GMT and then convert it to India time, the India time will be 53 hours and 30 minutes. The restriction does imply that any hour of a time to be printed takes at most two digits.

2. About the arguments h, m, and s in the constructor that you have to write. (a) No restriction is given on these parameters individually, so there is no need to check them individually. However, when the given time is translated to seconds, the result should be greater than –24 hours (ie. –1 day, or whatever that is in seconds) and less than 48 hours (i.e. 2 days, or whatever that is in seconds).

(b) The obvious way to translate h hours, m minutes, s seconds to a time in seconds is to convert all three values to seconds and add them together. Do this. This implies that h = 48, m = 0, and s = 1 is an illegal time, because it is greater than 2 days. Also, h = 48, m = 0, s = –1 is a legal time, since it is 1 second less than 2 days.

3. Time zones consist of capital letters. For example, "GMT" is a time zone but "gmT" is not. If you want to refer to a time zone, USE THE PUBLIC STTIC FINAL METHODS, like GMT, FOR THEM.

4. Extra (private) methods. You may write extra private methods. Whenever you are given a program to write, as you proceed, you may find it helpful to add new methods in order to keep other methods simple and understandable. You saw this in the program to anglicize integers, which we developed in class.

5. Rules for function toString. Function toString must satisfy the following.

(a) If the time is negative, then a 24 hour clock is to be used.

(b) Precede a negative time by '-'. Here are two times and their output from toString:

- 48370 seconds, i.e. (-13 hours, -26 minutes, -10 seconds) in GMT : "-13:26:10 GMT"

- 36970 seconds, i.e. (-10 hours, -16 minutes, -10 seconds in GMT : "-10:16:10 GMT"

(c) Precede a positive time by a blank. Here is a positive time and a negative time and the output from toString:

36970 seconds, i.e. (10 hours, 16 minutes, 10 seconds in GMT : " 10:16:10 GMT"

- 36970 seconds, i.e. (-10 hours, -16 minutes, -10 seconds in GMT : "-10:16:10 GMT"

(d) The hours, minutes, and seconds MUST each be two digits. Examples:

0 hours, 5 minutes, 16 seconds in GMT : " 00:05:16 GMT"

6 hours, 15 minutes, 6 seconds in GMT : "-00:15:06 GMT"

(e) The time 0 is midnight. It is usually thought of as AM, so print time 0 in 12-hour time with an AM indication. Similarly, noon is the beginning of PM, so it should be printed in 12-hour time with a PM indication.

It may be advantageous to write an auxiliary private function that, given a string that contains an unsigned integer, returns a string that represents the same unsigned integer but has length at least two (it prepends a leading 0 if necessary). Then call this function in several different places.