CS100J, Fall 2003, Assignment A3. Due on 03 October, before midnight.

About submitting assignments

Submit this assignment using the CS Course Management System:http://cms.csuglab.cornell.edu/. .**If CS100J is not listed, that means that you are not in our course list!** Email <shirk@cs.cornell.edu> from your Cornell account so that she has your Cornell netid. Explain that you are taking the course and want to be registered to submit assignments. Do this NOW --don't wait until it is time to submit the assignment.

Purpose

This assignment gives you practice with writing functions whose bodies have assignment statements, conditional statements, and blocks. You will also be working with one class.

Ground rules

You may work with one partner. If you do, get on the course management system several days before the assignment is due and follow directions for telling the system who your partner will be. Don't wait until the last minute to do this; do it several days before you want to submit the assignment.

This is (very, very roughly) a 10-hour assignment. Plan accordingly. Please keep track of your time and, just before you submit, tell us how many hours you took in a comment at the top of the class. Spend an hour reading this handout so that you thoroughly understand what we are asking for. Do this **with your partner** before you start programming!

We strongly suggest that you and your partner alternate writing the methods (but with the other person watching and helping). You will both benefit from this, even if you find it feels like it is taking longer.

Class `Time100J`

You will create several functions in class `Time100J`. To help you out, we give you a skeleton: a class with all the methods specified but with some stubs for bodies (empty body or simply a return statement), so that the program is syntactically correct and will compile. All the methods are specified fully. 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 specifications for the program. Get these from the course web page or from the CMS.

An instance of class Time100J 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)

India (IND) is included only to show that times are not always on hourly boundaries from GMT. Some time may appear negative or greater than 24 hours. This is because we allow a conversion of a time from one time zone to another, and a time of 0 hours GMT is -7 hours PDT (for example), 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 you 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, to get an overall view of what the class is for. For this purpose, use either the skeleton or, better, the JAVADOC spec for it.

This assignment will be presented as a series of tasks. **Finish each task completely before proceeding to the next**. And, 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 Interactions Pane, before proceeding to the next.

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. So, your first task is to complete getter methods getTime, getZone, and getAmPm. Once they are properly written, check them out! Make sure they work. This task should not be difficult.

Task 2: function toString.
Now, 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. Actually, to help you out, 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 isimplement these statement-comments. Of course, if you want, 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, with instances of the class for negative times, times in the range 0..24, and times greater than 24 hours. 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.
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 this function. Be sure to test it on ALL possible legal zones, as well as illegal ones, before proceeding to the next step. You can test it in two ways. First, make it public (temporarily only), so that you can call it from the Interactions pane. Second, notice that the second constructor calls isLegal, so, you can test isLegal using the 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?

Task 4: procedure setDisplay.
This procedure should be easy to write and test. It is a simple setter method.

Task 5: 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. Use the second constructor as an example in writing this one. Test it thoroughly before proceeding!

Task 6: function timeInZone.

This function is given a time in one zone and is asked to create a new Time100J instance 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 7: function compareTo.
Finally, you have to 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 8. 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).

What to submit

Use the link http://cms.csuglab.cornell.edu/ to get on the course website and submit the `Time100J.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. Remember that, as always, the name of a file is derived from the class it contains and that correct capitalization is required. Make sure you submit the .java file, and not the .class or the .java~ file!!!!!

By: David Gries, Jason Li