ICS314 Spring 2016 Calendar (.ics) assignment

Your team assignment is a straightforward implementation in Java using common software engineering tools. This is not difficult, but requires some attention to detail.

Why do a team project at all? Several reasons: First, it's in the catalog description for this class – it's that important. Second, working for a common goal, with people you don't necessarily know, is good experience. Finally, it's important to use some software engineering tools and techniques (including check-ins).

The temptation will be for one person to just do all the work. Yes, the assignment isn't very large, and there are usually complaints about working together. Team work was one of my least favorite things when I was an undergraduate.

One reason the project isn't large is that we just don't have time. In the real-world -- where a "small" project would be 100 KLOC -- it would be impossible for one person to do everything. We don't have that luxury in a single semester. This is a three-credit class, so 3 "lecture hours" plus 2-3 "study hours" a week. That's less than one 8-hour work day per week. A semester typically has 15 weeks, so that would be about 130 hours/semester if every moment of this class is about the team assignment. 130 hours is just over three work-weeks per person. How much can be done in 3 weeks on a new software project by new software engineers?

You see my point. So, instead of a project where you write code for the sake of writing code, we do a project where, yes, you need to write code, but there are other things involved. Like design, testing, user stories, configuration management… There will be a temptation to find code on the web, or use libraries. But that would be missing the point. Design and write and test your own code.

We will adopt the agile idea of two-week "sprints". You will have deliverables (including meetings with the TA) to make sure things are on track, then the finale/demo of what you've accomplished.

All of you have experience with using the Eclipse environment (http://www.eclipse.org/) to develop Java, so continue to use Eclipse. Host your software online (free) on git (https://en.wikipedia.org/wiki/GitHub). You must use revision control, and each team member must participate in committing code.

The task, outlined below, is to create .ics "event files", as described in RFC 5545. Of course, you won't be implementing everything that's in RFC 5545. I don't know of *any* commercial implementation of everything in RFC 5545, so you'll be doing a small piece. Your system will generate event files – *you are not building a system to read event files*: you can use an existing calendaring system like Outlook, Google Calendar, or Mac OS Calendar to read files (hint: that's a good way to test the output of your system).

The **first team deliverable** is worth 3 points and is due in Week 7**.**

Teams will have short (5-10 minute) meetings with the TA where you will demonstrate that:

- your team has registered your project on github
- everyone in the group has used a calendaring system of their choice to create a .ics event file and has looked at the content of that file with a text editor such (such as vim or textedit). "Reverse engineering" .ics files can offer significant insight.
- your team has functioning code, no matter how minimal: You must demonstrate code that generates a .ics file for a simple, single occurrence event such as: Study for exam, 1pm-2pm HST, 12 May 2016, in Hamilton Library. You must then demonstrate that the calendar program of your choice (say, google calendar) successfully reads and processes the .ics files you generate. The code could be a simple as print statements, but for that single event, but it must generate an .ics file that is readable by a calendar project.
- In the real-world, your code would be embedded in a larger calendaring system, so don't worry much about a user interface. In fact, to create .ics files your user interface can be command line or simple drop-down menus.

The **second team deliverable** is worth 5 points and is due in Week 9**.**

Teams will have short (5-10 minute) meetings with the TA where you will demonstrate that:

- system allows the use of the "geographic position" field described in section 3.8.1.6 of the RFC. Events are not required to use the field, but if the information is provided by the user, your system must support it by putting it into the event file.
- your systems must also implement the "classification" field as described in section 3.8.1.3 of the RFC. Events are not required to use the field, but if the information is provided by the user, your system must support it by putting it into the event file
- Your team has a systematic way of testing the code *as you are writing it*. I suggest, among other things, using JUnit to develop and execute test cases as you are developing code (http://www.junit.org/node/49)

The **third team deliverable** is worth 5 points and is due in Week 12**.**
Teams will have short (5-10 minute) meetings with the TA where you will demonstrate that:

- Your system can read-in an arbitrary number of event files for a single date, sort of the events by start time, then compute the great circle distance (https://en.wikipedia.org/wiki/Great-circle_distance), based on the geographic position of successive events. You shall record the great circle distance in the comment field of the first-occurring event (see section 3.8.1.4 about comments). Here's an example:

EventN occurs from 11am-1pm
EventNplusOne occurs from 130pm-3pm
EventNplusTwo occurs from 4pm-5pm

Your system will compute the great circle distance, in statue miles and kilometers, from the location of EventN to EventNplusOne, and record that in the comment field of EventN.

Similarly, your system will compute the great circle distance, in statue miles and kilometers, from the location of EventNplusOne to EventNplusTwo, and record that in the comment field of EventNplusOne.

The idea is to give the user an idea how far apart their events are (can they walk? Bus? Drive? Fly?)

But I said that location was optional for events. What if there is only one event? What do you do with the last event of the day? What are you going to do then? You'll have to think about it and come up with something reasonable.

The **final deliverable (demo)** is due no later than in **29 April** in Week 15. The deliverables are:

- The URL for a 5 minute *narrated* video demonstration of your final project. One demo per team. *Your demo shall not be longer than five minutes.* You can use jing.com, screencast-o-matic.com, …., or if you are using a Mac, QuickTime, to capture your screen to video. You need to demonstrate the functionality described below.
- A one-page pdf document convincing me that your system is correct. That is, how did you test your system and what were the results?
- A link to your documented source code on github. We will check that commit history to make sure everyone on the team is participating.

Make sure I can view your video demo and access your code (and see the history of your commits).

Summary and some hints
You are developing a stand-alone application to create .ics event files. You must demonstrate the following are implemented:

- Version (section 3.7.4 of RFC 5545)
- Classification (3.8.1.3)

- Comment (3.8.1.4) Your system will compute the great circle distance between events and store it in the comment fields.
- Geographic position (3.8.1.6)
- DTSTART (3.8.2.4)
- DTEND (3.8.2.2)
- Time zone identifier (3.8.3.1)

Start small with the simplest .ics file you can generate, test, and iterate.
Everyone needs to be helpful (to other teams if necessary), patient, and not wait to the last minute. Note that Section 4 of the RFC has several examples of .ics files.