Bonus Points FTW! - ELSA (OPTIONAL Sprint 6)

Due Tuesday, May 2 at 8 a.m.

CSE 1325 - Spring 2023 - Homework #13 / Sprint 6 / Rev 0

IMPORTANT: This is an OPTIONAL assignment for those who would like to earn extra credit by implementing additional, non-required features.

- To submit this assignment, commit and push your code to the cse1325/P13 directory *with an accurate Scrum spreadsheet* by the deadline.
- To skip this assignment, do nothing. Your grade will not be affected.

Assignment Background

The Exceptional Laptops and Supercomputers Always (ELSA) store offers the coolest (ahem) deals in computing technology for the savvy computer geek and their lucky friends. Each computer can be hand-crafted to match the technologist's exact needs, with a growing selection of convenient predefined configurations already purchased by your discerning peers (and competitors). They now belatedly seek to automate their physical and online storefronts, replacing paper forms and ink pens with the miracle of modern computing technology. Your goal is to prove that you can implement their store management system and thus win the contract to build it, with all of the associated fame and cash.

This is OPTIONAL Sprint 6 of 6.

IMPORTANT: Do NOT use full_credit, bonus, or extreme_bonus subdirectories for this project. Instead, organize your code into two packages, `store` and `gui`. ALWAYS build and run from the P13 directory.

The Scrum Spreadsheet

The Feature backlog has not changed for this sprint.

The intent of using a *very* simplified version of Scrum on this project is to introduce you to the concept of planning your work rather than just hacking code at the last minute and hoping for the best. **Professionals make a plan and then execute it.** Amateurs sling code and suffer the consequences.

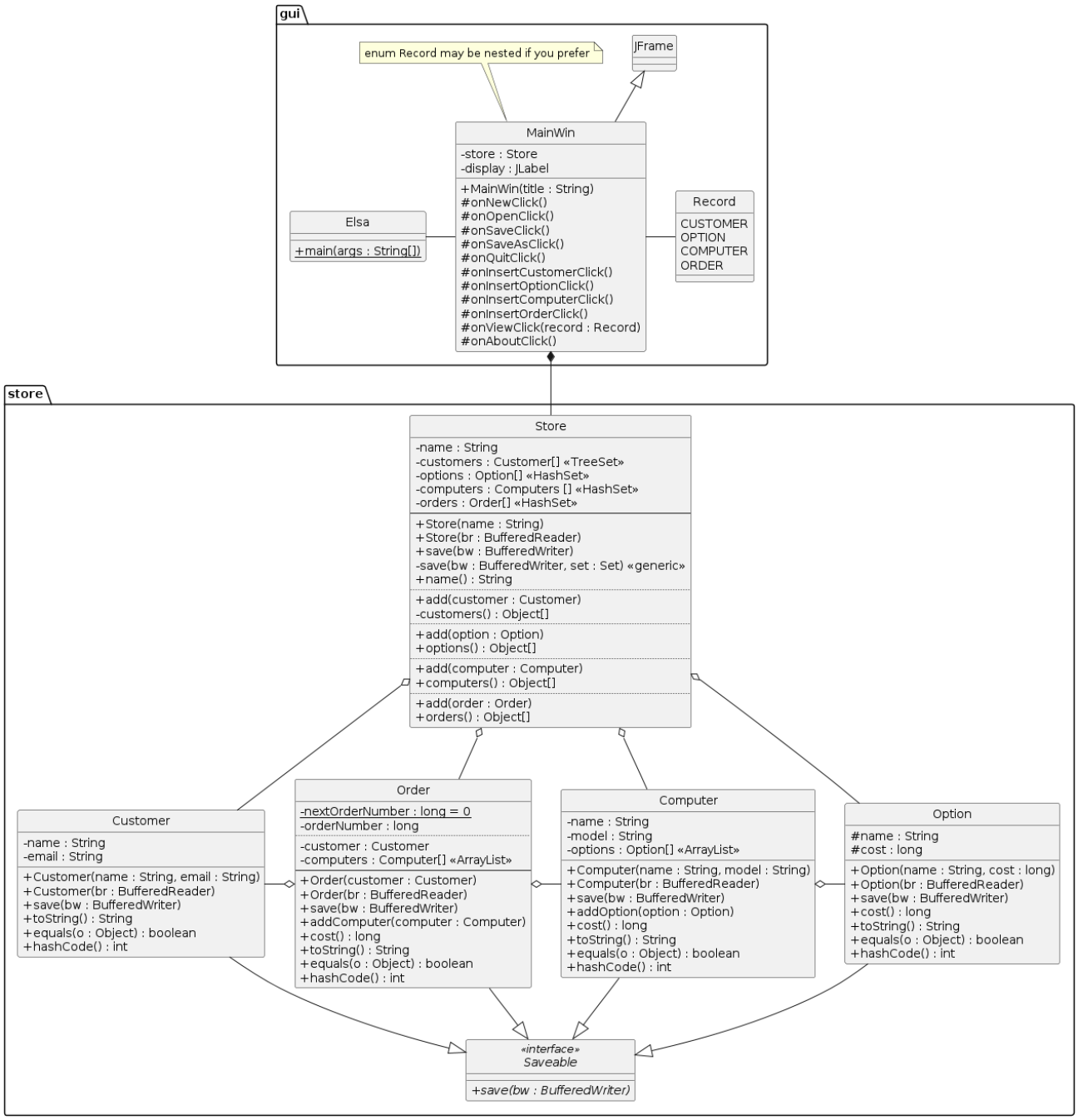
Submitting an *accurate* spreadsheet is part of your grade for these assignments. This is not what you *wish* had happened - it's what actually happened. **Be certain you update the spreadsheet and add, commit, and push it at cse1325/P13/Scrum.xlsx before the end of the sprint!** If you prefer to use an online spreadsheet, ensure that it is publicly visible and include a cse1325/P13/Scrum.url text file instead containing its *public* link.

For this sprint, at a minimum, you will update the Product Backlog AND Sprint 06 Backlog tabs. Then add, commit, and push your Scrum spreadsheet.

Now that you *have a plan* - on to coding!

Class Diagram

Because the features you choose to implement may vary, the class diagram has not been updated for this sprint. The diagram on the next page covers through Sprint 5. Dotted lines are purely visual and have no semantic meaning.



Overview of Changes

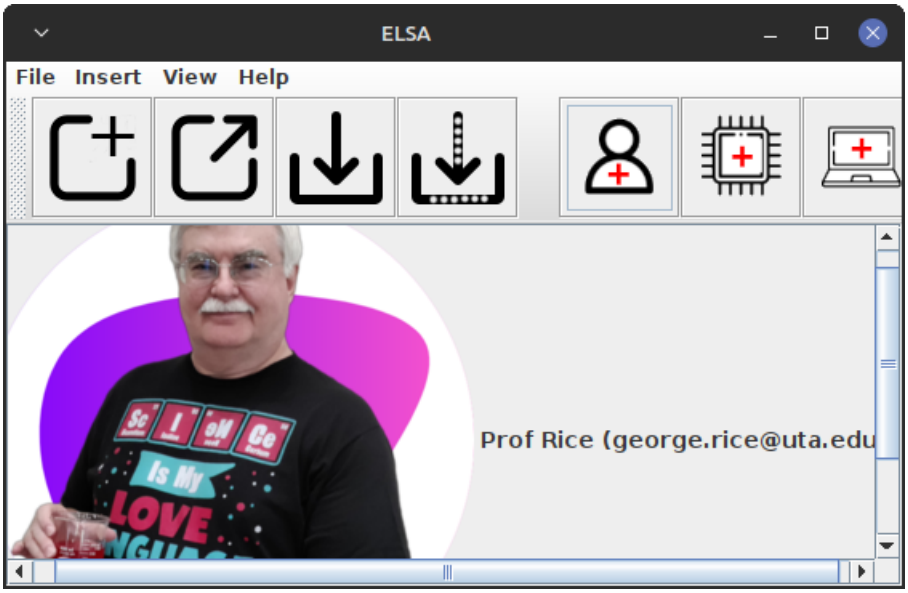
The Scrum spreadsheet lists several optional features that you may implement along with the additional points gained for a *competent and complete* implementation. An imperfect implementation may garner fewer points, while true excellence may garner quite a bit more.

I give general guidance for some of the optional features below, but it is up to you to work out the best approach.

Feature SCROL

Swing provides a JScrollPane widget that will enable scrolling of a lightweight component such as a JLabel in a smaller space. Just put your display in the scroll pane and add that instead of display.

You can find additional guidance at
https://docs.oracle.com/javase/tutorial/uiswing/components/scrollpane.html



Feature DEPR

You'll get the full points for implementing this feature for just one of the classes Option and Computer, with additional points for supporting both.

Deprecating Options and Computers requires adding a Boolean field such as `deprecated` to each class with a getter and a setter. The default value on constructing a new object is, of course, `` false``.

For Store.options() and Store.computers(), rather than returning the toArray() value of the field, you will need to iterate over the fields and construct a new Set containing only *non-deprecated* items. Then return the toArray() of that Set instead.

It's perfectly OK to continue to reference deprecated Computers in existing Orders, and (for purposes of this project, at least) deprecated Options in existing Computers (although deprecating existing Computers that contain a deprecated Option would be rather nice). But *new* Computers can't be defined with deprecated Options, and Orders can't be created for deprecated Computers.

Feature SBAR

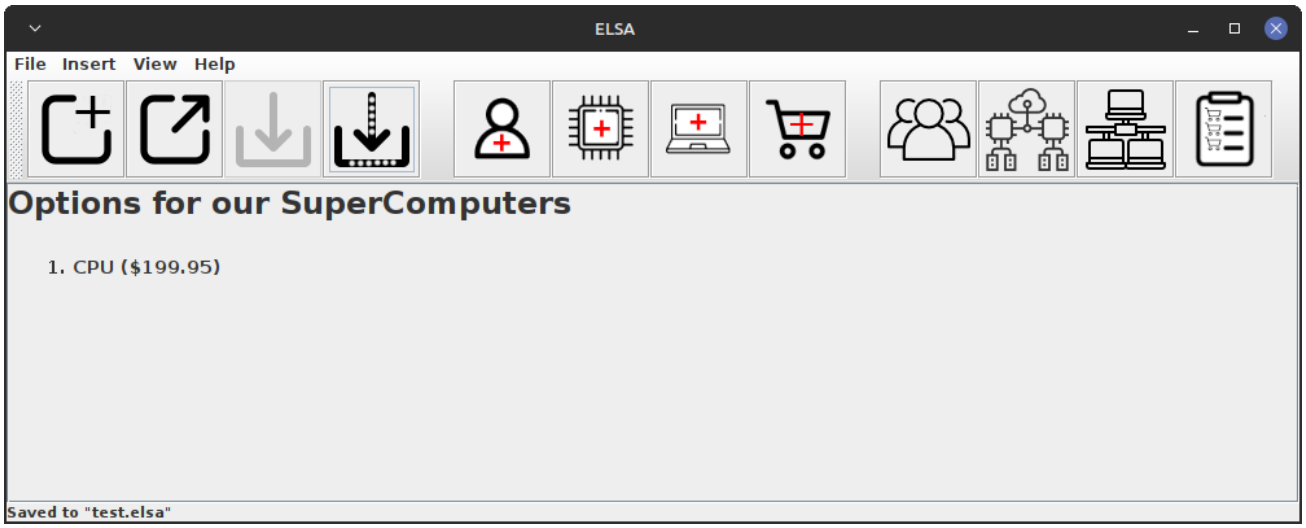
A status bar could be a simple as a JLabel added to BorderLayout's page end area. I recommend that you add a method such as setStatus(String) for changing the status bar's text.

At the *start* of each listener, call setStatus("") to clear the status bar. You don't want old status hanging around while the user is creating new data. For View and Help > About, I would just leave the status blank.

After creating new data successfully, report this, for example "Created Customer " + customer.

After opening or saving a file (for example) report success or failure, for example "Opened " + filename.getName() or "Failed to open " + filename.getName().

For other events of note, use your own judgment.



Feature DTAIL

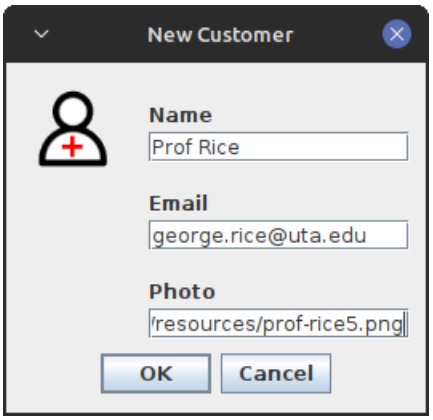
You'll get the full points for implementing this feature for just one of the classes Customer, Option, and Computer, with additional points for supporting two or all three of them.

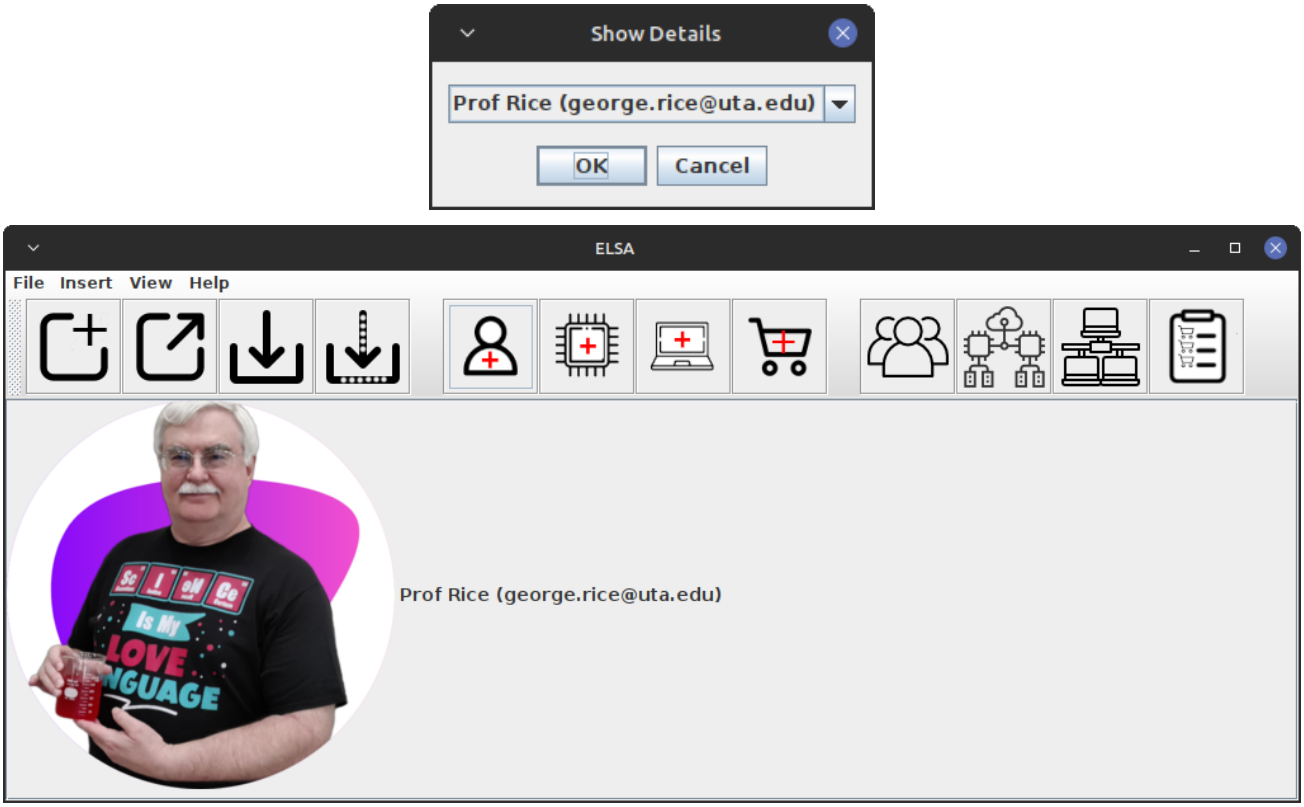
Each of the 3 data classes - Customer, Option, and Computer - may need an additonal String field (photo or image, for example) to hold the filename of the graphic to be included with the details. The toString() method need NOT change, but you'll need a method that MainWin can use to get that filename! Also be sure to save and reload the image filename (you needn't do anything to save the images themselves - we'll assume that's handled elsewhere).

In MainWin, your common unifiedDialog method (if you wrote one) will be really handy here: Just add an "Image" field to the unifiedDialog call and add that to the constructor parameter list for each object. Otherwise, you'll need to add this text field or a FileChooser button to your custom dialogs. (Using a FileChooser dialog would be worth additional bonus points, of course.)

In MainWin, then, you will need (for example) a View > Details menu item, which would allow the user to select Customer, Opiton, or Computer (if you implement all three), then select the record of interest from a JComboBox dialog, and finally put the information into the main window.

Your data area is probably a JLabel, which conveniently enough will display both an ImageIcon and text. Be sure to display.setIcon(null) for your non-detailed display selections!





Feature SPEC

Add a subclass to Option, for example Cpu or Ram, that includes an additional field such as gigaHz or gigaBytes. Override toString(), equals(), and hashCode() to include the new field.

Provide a means to create this specialized Option - for example with a radio button selector for "CPU" or "Other". For a "CPU" then, add an additional field to collect data for the additional field. The Cpu objects are stored in the options Set along with the Option objects, however.

The remaining challenge is saving and restoring these specialized Option objects. The challenge is that the Store constructor must know *in advance* (for each object in options) whether to construct an Option or a Cpu object.

My usual approach introduces a slight bit of assymetry:

- In the Option save method, *first* write "Option" to a line, then the local fields to a separate line each.
- In the Cpu (or other subclass of Option) save method, *first* write "CPU" (or other unique name) to a line, then the superclass fields to separate lines, and finally the local field to a line.
- In the Store.open method when loading Option object, *first* read the unique name for the class to be instanced, and then use switch or if/else to invoke the correct constructor.

The alternate approach avoids this assymetry at the expense of additional code elsewhere: Create a separate Set in Store for each subclass. Store.save and the Store constructor then treat this in every way as separate data types. However, Store.options() must return the *union* of the options Set and each Set containing subclasses of Option.

The latter approach is more symmetrical but requires a bit more code. Either should work for you.

Other Features

If you have ideas for other features you would like to implement, please contact the Professor *first*. If the feature is appropriate, I will try to estimate the work required to implement it and base the bonus point value on that estimation.