Viewing the Goods - ELSA (Sprint 2)

Due Tuesday, March 7 at 8 a.m.

CSE 1325 - Spring 2023 - Homework #6 / Sprint 2

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 Sprint 2 of 6.

IMPORTANT: Do NOT use full_credit, bonus, or extreme_bonus subdirectories for this project. **Instead, organize your code into two packages,** `store` (last sprint's code) and `gui` (added this sprint). Remember, packages are subdirectories, so store last week's code in `cse1325/P06/store` and this week's code in `cse1325/P06/gui`.

The Scrum Spreadsheet

> **IMPORTANT: The Feature backlog HAS CHANGED for this sprint!** Please update your spreadsheet from the one provided in the Canvas Assignment page.

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/P06/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/P06/Scrum.url text file instead containing its *public* link.

For this sprint, at a minimum, you will update the Product Backlog AND Sprint 02 Backlog tabs, similar to your approach in Sprint 1. If you have questions, check the Requirements for Sprint 1 and then if necessary ask us via email. We love questions!

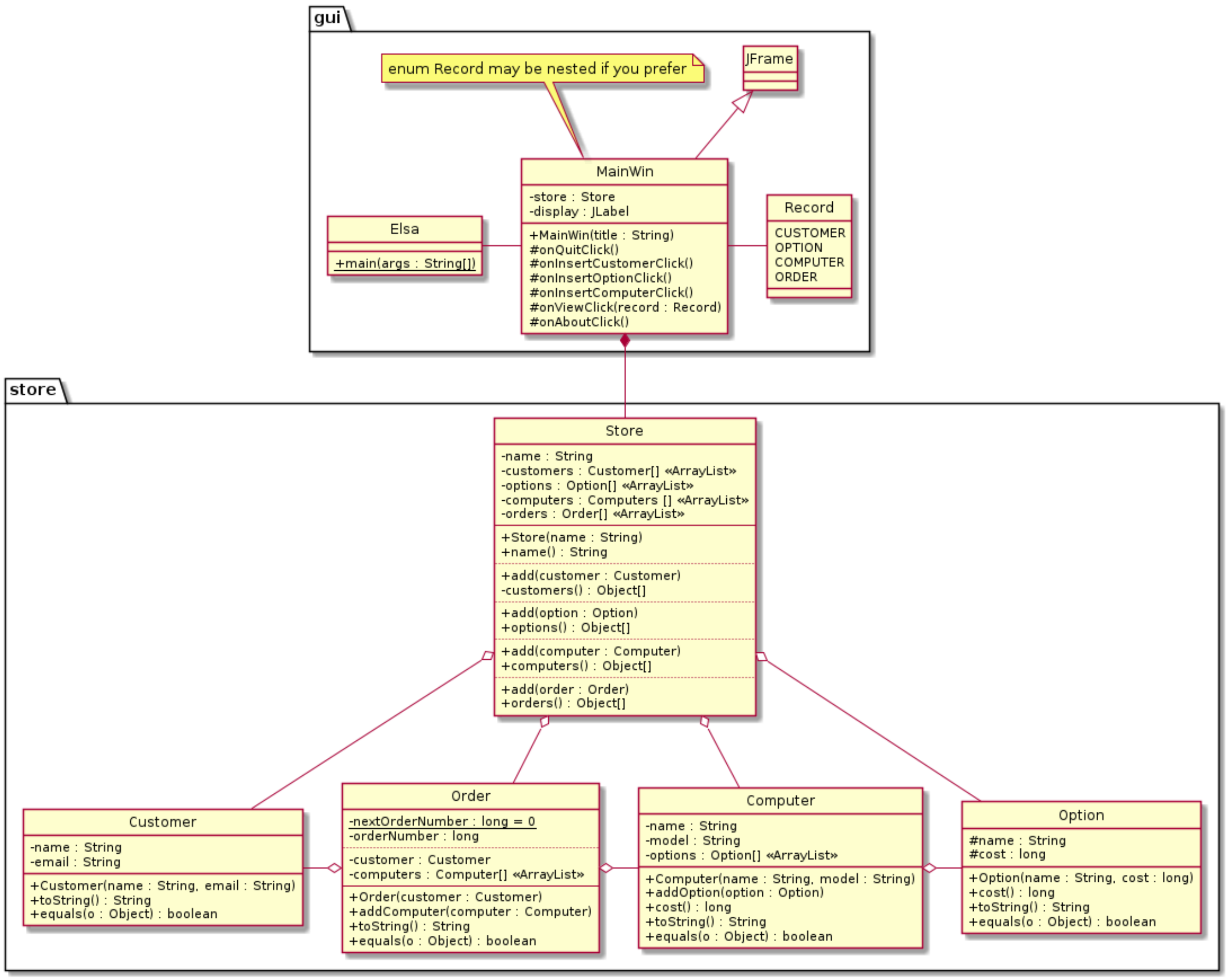
Now add, commit, and push your Scrum spreadsheet.

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

Class Diagram

The diagram on the next page covers just Sprint 2. A more complete diagram will be provided once you have a basic understanding of Graphical User Interfaces (GUI) and Java Swing (our GUI toolkit). And updates to fix bugs and design issues are all but inevitable. (Dotted lines are purely visual and have no semantic meaning.)

For this sprint, we will move our Store-related code into package `store`, then focus on a basic implementation of package `gui` based on the Nim code from Lecture 12. Details instructions follow the class diagram - read and follow them *closely* to avoid wasting time and writing incorrect code!



Write the Code

Create Package store

Copy `cse1325/P05` to `cse1325P06`. The bash command for this is `cd cse1325` and then `cp -r P05 P06` (the `-r` flag means "recursive", or copy all of the P05 subdirectories all the way down to the P06 destination).

Now create a `store` subdirectory in P06 (`cd P06` and then `mkdir P06`) and move all of the Java files into it (`mv *.java store`).

Add `package store;` as the first line of all of your source files in the `store` directory.

Finally, build and test last sprint's code. Remember that you MUST build from the `cse1325/P06` directory, which is where we left your `build.xml` file. So, from `cse1325/P06` you would build your code (`ant`) and then run it (`java store.TestStore`). You should see exactly the output that you saw when you finished P05.

If not, debug it until it does. If you get stuck, let us know. We love questions!

Baseline Package gui

We now need to add a Graphical User Interface to our ELSA store. While you may write your MainWin.java code from scratch if you insist, I *strongly* recommend that you start with an existing GUI application: Nim from Lecture 12. (We call this a *baseline*: An existing body of code reasonably known to work and from which a new system may be built.)

Copy the complete Nim application from Lecture 12 into a new `gui` subdirectory of `cse1325/P06` (something like `cp ~/cse1325/12/code_from_slides/nim gui` may work, or use your graphical file manager).

Now clean up the files in your gui subdirectory.

- Delete the `build.xml` file - you will ALWAYS build from cse1325/P06! (`cd gui` and then `rm build.xml`).
- Delete all of the images - you'll need new ones for our project (`rm *.png`).
- Delete the UML diagram (`rm *.xmi`). I will provide the diagrams in the requirements (see above).
- Delete the Nim and AboutDialog classes - we won't need them (`rm Nim.java` and `rm AboutDialog.java`).
- Rename `PlayNim.java` to `Elsa.java` - this will contain our main method (`mv PlayNim.java Elsa.java`).

This is a GREAT time to add / commit / push your code to GitHub!

Write the Main Method in class Elsa

Edit `gui/Elsa.java`.

- Declare the package at the top.
- Fix the class name to match the filename.
- Change the title of the main window, for example to "ELSA". This will be the *default* title, although eventually the title will contain the name of the ELSA file we are using in our application.

Write the Main Window in class MainWin

Edit `gui/MainWin.java` and declare the package at the top.

Constructor: M E N U

Create a Record enum that lists the 4 types of classes we have currently defined and may want to view in the main display area - CUSTOMER, OPTION, COMPUTER, and ORDER. This can be public and in a separate file, a package-private enum in `MainWin.java` but outside the class, or in class scope, whichever you prefer. (You needn't follow the class diagram in this.)

Create the following menu structure. Carefully follow the pattern established in Nim. If you aren't sure about something, ask!

- File > Quit: Unchanged, calls onQuitClick().
- Insert > Customer: Calls onInsertCustomerClick().
- Insert > Option: Calls onInsertOptionClick().
- Insert > Computer: Calls onInsertComputerClick().
- View > Customers: Calls onViewClick(Record.CUSTOMER).

- View > Options: Calls onViewClick(Record.OPTION).
- View > Computers: Calls onViewClick(Record.COMPUTER).
- Help > About: Calls onAboutClick().

Note the reuse of the onViewClick method for 3 different menu items. This is common in GUI applications.

IMPORTANT: We omit creating Orders for now to reduce the complexity of this sprint, but it is expected in be allocated to sprint 4. If you have time and would like to get ahead, once you've finished the other features for this sprint, come back and implement inserting and viewing Order objects as well.

Constructor: T O O L B A R

I recommend that you comment this portion out for now. Get your menu-driven application running first, and add the toolbar at the end.

Constructor: D I S P L A Y

You may remove "S T I C K S" from the comment for the subsection header. We're doing Computers now!

Set up the display for data (such as lists of computers or customers) as you please based on the Nim example.

Instead of instancing a new Nim game, instance a new Store.

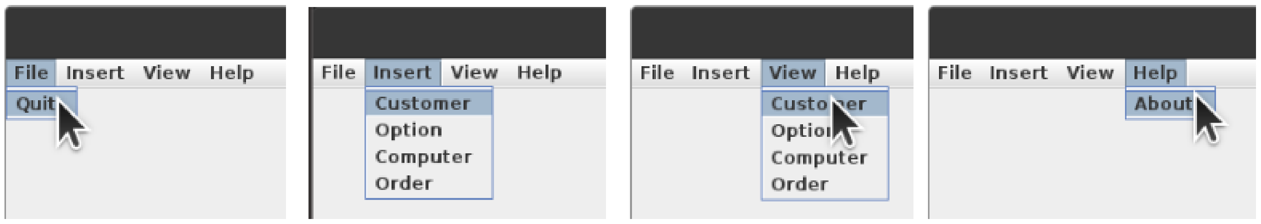
Empty Listeners

Start with a complete set of Listeners with empty bodies. We just want to get our GUI up and running first, then we'll add bodies to the listeners to make our store work!

Build and test all of your code thus far.

How does it look? Any changes needed? Yes, the About dialog still says Nim (but with no logo). We'll fix that below.

Get it right otherwise, then on to writing the listeners!

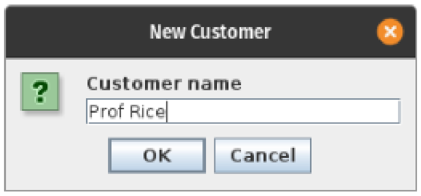


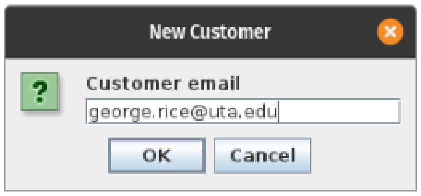
Write the Listeners

You will receive full credit for this assignment for providing (for example) two dialog boxes when inserting a new Customer. More advanced students should remember that a later sprint will require a single "unified" dialog box to collect both the Customer name and email address along with unified dialogs for the other three record types. So if you would like to save time later and can figure out how, consider writing the unified dialogs now.

Customer

Select a JOptionPane dialog that returns a String entered by the user, and use it twice to obtain the Customer's name and email address. Supply that to the Customer constructor and pass the new Customer to Store's add method.





Don't forget a try / catch in case the user enters an invalid email address or clicks Cancel. Don't abort the application on an exception. It would be very nice to provide a warning message dialog if the email address was invalid, but no warning email address if the user clicked Cancel, altough points won't be deducted if you don't. Do you see how to easily accomplish this?

Option

Option needs a String for the name but a long for the cost. The simplest way to obtain the cost is to get a String from the user and convert it to a double (since they will enter a decimal point), then multiply by 100 and cast to a long. This will get you full credit.

But feel free to instance a different, more suitable widget to use as the message in a confirm dialog. If the confirm dialog indicates that OK was selected, get the cost from that widget instead.

Computer

The constructor portion looks a lot like Customer: Get two Strings for Computer name and model, and you can construct a Computer.

But you need more - you need *Option* selections as well. We can obtain an Object[] array of Option objects from store, but how to select one?

The best way is to instance a JComboBox, passing the Object[] array you retrieved from `store` as the constructor parameter. Then pass the JComboBox object as the `Object message` parameter of JOptionPane's `showConfirmDialog`.

- If the button number returned indicates the user selected an Option, use JComboBox's `getSelectedItem()` method to obtain it and add to the Computer object.
- If the button number returned indicates the user is finished selecting Options, pass the Computer object you've built up to store's `add` method.

It's a new Computer to order!

(If you opt to come back and insert Orders as well, it's fairly similar to Computer. The user should first select a Customer using a JComboBox just like the one in Computer, instance the Order, and then add Computer instances following the same pattern as Computer. It's no harder - just more code, so it was moved to a later sprint.)

View Menu

Because each getter in Store returns Object[], the onViewClick method can be shared by all.

Define a header (such as for "Customers") and Object[] array (for the array returned by a getter), then use if/if else or switch to assign objects to those variables based on the parameter.

Given that, use plain text directly (see the Hint section for handling newlines) or simple HTML formatting to create your header and then a numbered list of the Objects returned by the store.

Here's an example HTML document that you may adapt *if you like* to implement your onViewClick (but feel free to be as creative as you like!):

```html
<html>
  <p><font size=+2>My Header Text</font></p>
  </br>
  <ol>
    <li>First item in the numbered list</li>
    <li>Second item in the numbered list</li>
  </ol>
</html>
```

Quick HTML tutorial:

- `<html> ... </html>` defines that you will use HTML to define rich text
- `<p> ... </p>` defines a paragraph
- `<font size=+2> ... </font>` specifies the size of the text relative to the default
- `</br>` is a newline
- `<ol> ... </ol>` is an "ordered" (numbered) list
- `<li> ... </li>` is a list item within the ordered list

The display JLabel has a setText method to *overwrite* (NOT append) what is displayed. If you need to append, first use getText to get what's currently displayed, append to it, and the use setText to overwrite what is displayed.

Toolbar

The toolbar looks and works very similar to what we had in Nim, so uncomment the Nim code and adjust to support 3 insert behaviors and 3 view behaviors using the same listeners as we used for the menu. (We don't have "new Store" yet, and the exit button is optional.)

You may draw your own button icons, or select from (and customize, if you like - I usually do) icons from the image libraries provided in the lecture. Be careful to follow the license. I usually prefer 64x64 or sometimes 32x32 pixel icons, but your tastes may vary. Swing will NOT scale your icons for you, so use an external tool such as the Gimp or an online image editor.

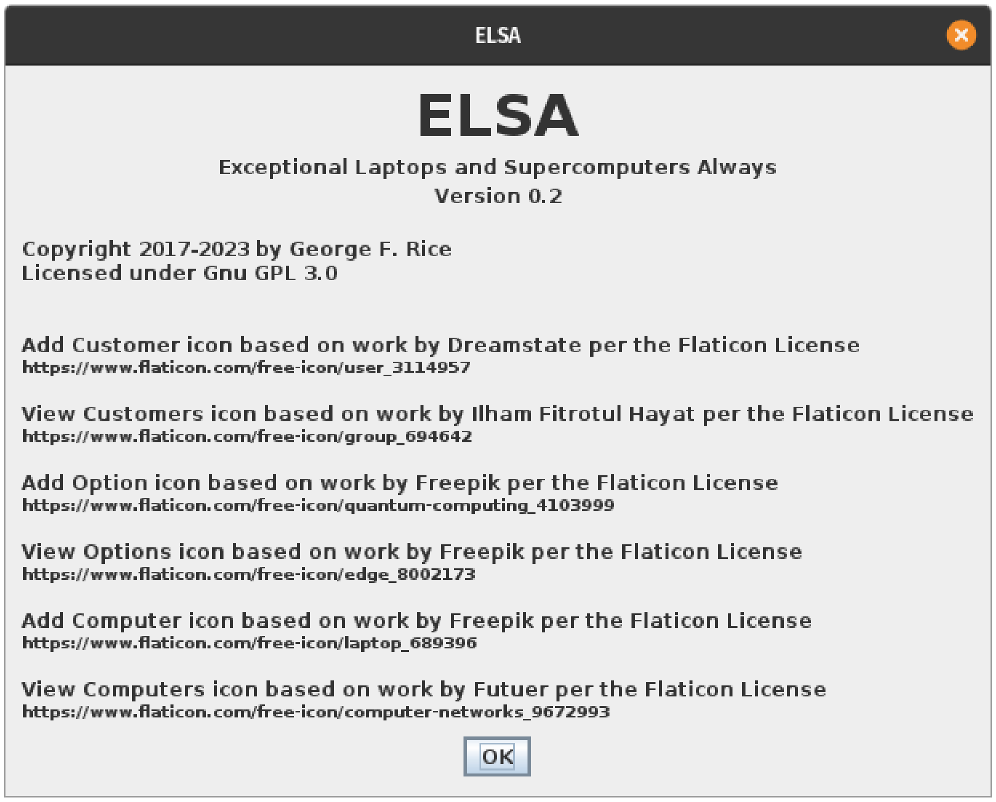
Professional artists (NOT me) talk about a "design language" for an icon set. For example, I include a red "+" for each "insert" icon, and my view icons contain 2 or more of the "insert" icons without the "+". But I may be doing it wrong. When you turn pro, hire good graphic artists and *pay them well*. Your application will look richer, and you will be, too!

As with classes in packages, loading an icon requires a path *from P06*. If you store an icon in `cse1325/P06/gui/resources/add_customer.png`, you would load it using
`ImageIcon ii = new ImageIcon("gui/resources/add_customer.png");`

About Dialog

If you select icons from a library *and the license requires*, you'll need to adapt the Nim onAboutClick method to support ELSA and meet your license obligations - a logo is NOT required, just text. If not, you may wait until the next sprint, when it will be required with some jazzy drawings.

I set the version number to 0.2 because this is Sprint 2 and that's easy to remember. In general, version numbers are of the form Major.minor.patch, where major increments when backward compatibility break, minor increments when new features are added, and patch increments with each bug fix release. But some projects use a different scheme, so do whatever you like for your ELSA project!



ELSA

Exceptional Laptops and Supercomputers Always
Version 0.2

Copyright 2017-2023 by George F. Rice
Licensed under Gnu GPL 3.0

Add Customer icon based on work by Dreamstate per the Flaticon License
https://www.flaticon.com/free-icon/user_3114957

View Customers icon based on work by Ilham Fitrotul Hayat per the Flaticon License
https://www.flaticon.com/free-icon/group_694642

Add Option icon based on work by Freepik per the Flaticon License
https://www.flaticon.com/free-icon/quantum-computing_4103999

View Options icon based on work by Freepik per the Flaticon License
https://www.flaticon.com/free-icon/edge_8002173

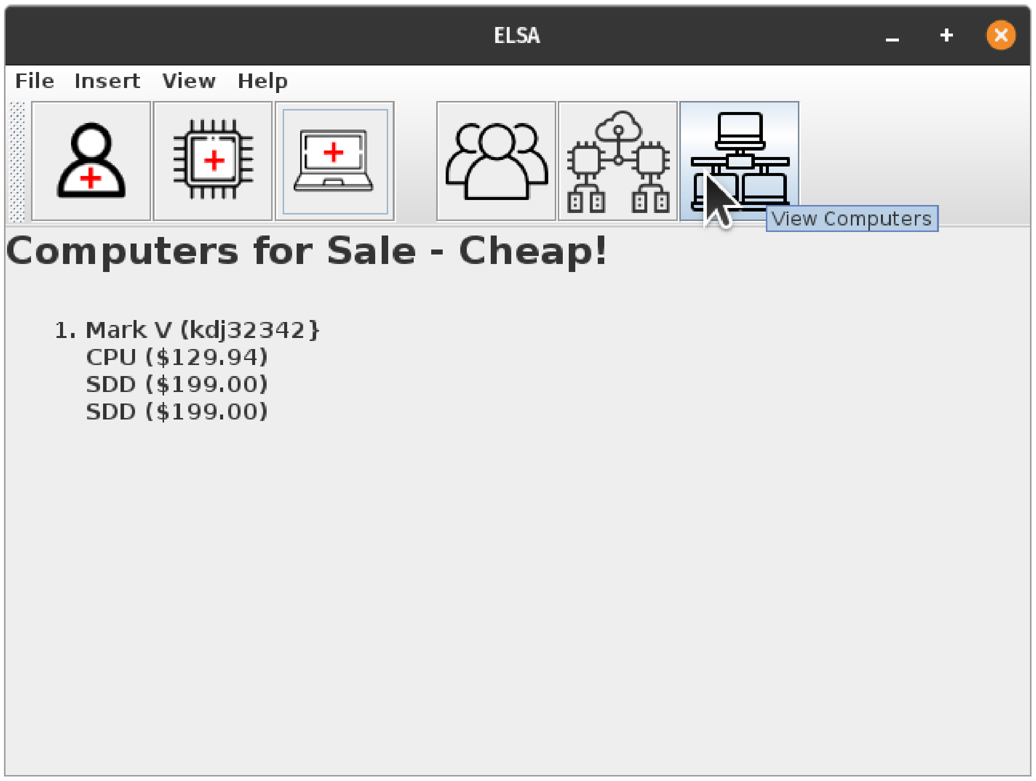
Add Computer icon based on work by Freepik per the Flaticon License
https://www.flaticon.com/free-icon/laptop_689396

View Computers icon based on work by Futuer per the Flaticon License
https://www.flaticon.com/free-icon/computer-networks_9672993

Main Window

The suggested solution's main window looks like this. You do NOT need to follow these images closely: Make ELSA your own application! This is just to provide some guidance as to what we are expecting to see in general.

If you don't implement ALL of the features, implement the ones you can. Since this is the longest assignment of the semester, we will grade as gently as possible.



Computers for Sale - Cheap!

1. Mark V (kdj32342}
   CPU ($129.94)
   SDD ($199.00)
   SDD ($199.00)

Hints

These are "nice to have" features, for which you will NOT lose points if you skip them. But students frequently ask how to implement them.

- To put text in the upper left corner of the JLabel display in the main window, try
- To change all `\n` (from a toString()) into `<\br>` so they show up in the JLabel, try

```
s.replaceAll("<","&lt;")
 .replaceAll(">", "&gt;")
 .replaceAll("\n", "<br/>")
```

This saves all non-HTML < and > characters, then changes `\n` to its HTML equivalent `<br/>`.