Saving Private and Draw On - ELSA (Sprint 3)

Due Tuesday, March 21 at 8 a.m.

CSE 1325 - Spring 2023 - Homework #7 / Sprint 3

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 3 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 P07 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/P07/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/P07/Scrum.url text file instead containing its *public* link.

For this sprint, at a minimum, you will update the Product Backlog AND Sprint 03 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!

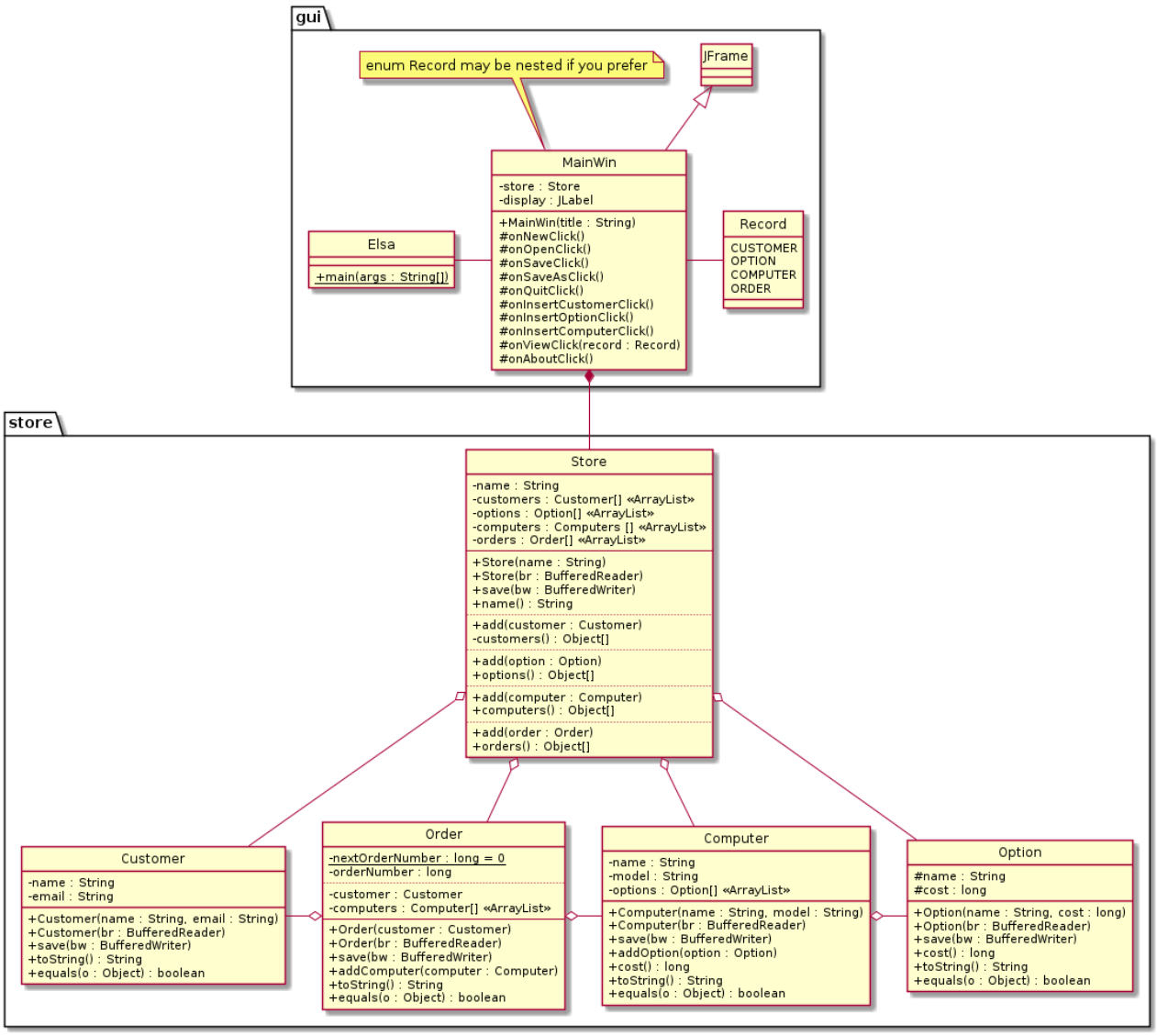
Now add, commit, and push your Scrum spreadsheet.

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

Class Diagram

The diagram on the next page covers just Sprint 3. 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

Updating Package store

Each class in package store will get two new members, a save method that writes *that object's data* to a BufferedWriter stream created by MainWin, and a constructor that reads the data back using a BufferedReader stream created by MainWin. **We will only grade saving and loading the Customer, Option, and Computer data for this sprint, NOT the Order data,** although you may save and recreate Order data if you would like to get ahead on sprint 4.

Save Method

The save method always has the declaration
`public void save(BufferedWriter bw) throws IOException` regardless of your class name. Write this method *before* the constructor (although as usual, its position in the class is irrelevant).

This method must write all of its class' fields to the `bw` stream, a BufferedWriter object, one line per field. This includes both private or protected scalar fields like `String name` or `long cost`, instances of your own classes such as `Customer`, ArrayLists like `ArrayList<Option> options`, and instances of Java Class Library classes such as `Color` (you probably have none of these yet).

- To write a String field such as `String s`, you would use `bw.write(s + '\n');`
- To write any primitive field such as `int i`, you would use `bw.write("" + i + '\n');`
- To write an instance of one of your own classes such as `Option o = new Option("CPU", 29995)`, you would simply use `o.save(bw)`.
- To write an ArrayList such as `ArrayList<Option> options`, you must first write the number of elements in the ArrayList using `bw.write(options.size() + '\n');`. Then, write each element in the ArrayList using a for-each loop, `for(Option option : options) option.save(bw);`
- To write an instance of a Java Class Library object such as `Color c = new Color(0xffbb00);`, you must find a matching method / constructor set such that the getter method returns a primitive that one of its constructors can use to recreate it. For `Color`, for example, we have an `int getRGB()` method that returns an int that can be accepted by `new Color(int rgb)` to recreate that object. So we would use `bw.write("" + c.getRGB() + '\n');`

In each case, you should be able to tell that we've written out the fields in the class, one per line, which will make it easy to retrieve them for use in constructing a replacement class when opening the file.

Constructor

The constructor always has the declaration
`public ClassName (BufferedReader br) throws IOException` where `ClassName` is the name of your class. Write this constructor *after* writing the `save` method (although as usual, its position in the class is irrelevant).

This method must read all of the class' fields back in from the `br` stream, a BufferedReader object, *in exactly the same order as they were written out.*

- To recreate `String s`, you would use `s = br.readLine();`
- To recreate `int i`, you would use `i = Integer.parseInt(br.readLine());` Similarly, you would use `l = Long.parseLong(br.readLine());` for a long,
  `d = Double.parseDouble(br.readLine());` for a double, and so on.
- To recreate an instance of one of your own classes such as `Option o`, you would simply use `Option o = new Option(br);`
- To recreate an an ArrayList such as `ArrayList<Option> options`, you must first read the number of elements using `int size = Integer.parseInt(br.readLine());`. Then, recreate and add each element in the ArrayList using a while loop,
  `while(size-- > 0) options.add(new Option(br));`
- To recreate an instance of a Java Class Library object such as `Color c`, use the constructor that complements the getter you selected, converting the String read from the file to the necessary primitive, for example, `Color c = new Color(Integer.parseInt(br.readLine()));`

Enhanced Logo

Rather than just a static image as the logo in your Help > About dialog box, also use a JPanel subclass which we usually call Canvas to draw a simple (or complex, if you prefer) pattern behind the logo (see Lecture 15). This will give you a little drawing experience before the exam.

You may or may not need a constructor for your Canvas subclass depending on the sketch you plan to draw.

Overriding the `Dimension getPreferredSize()` will enable you to specify the intended width and height of the Canvas, although Java may have other ideas.

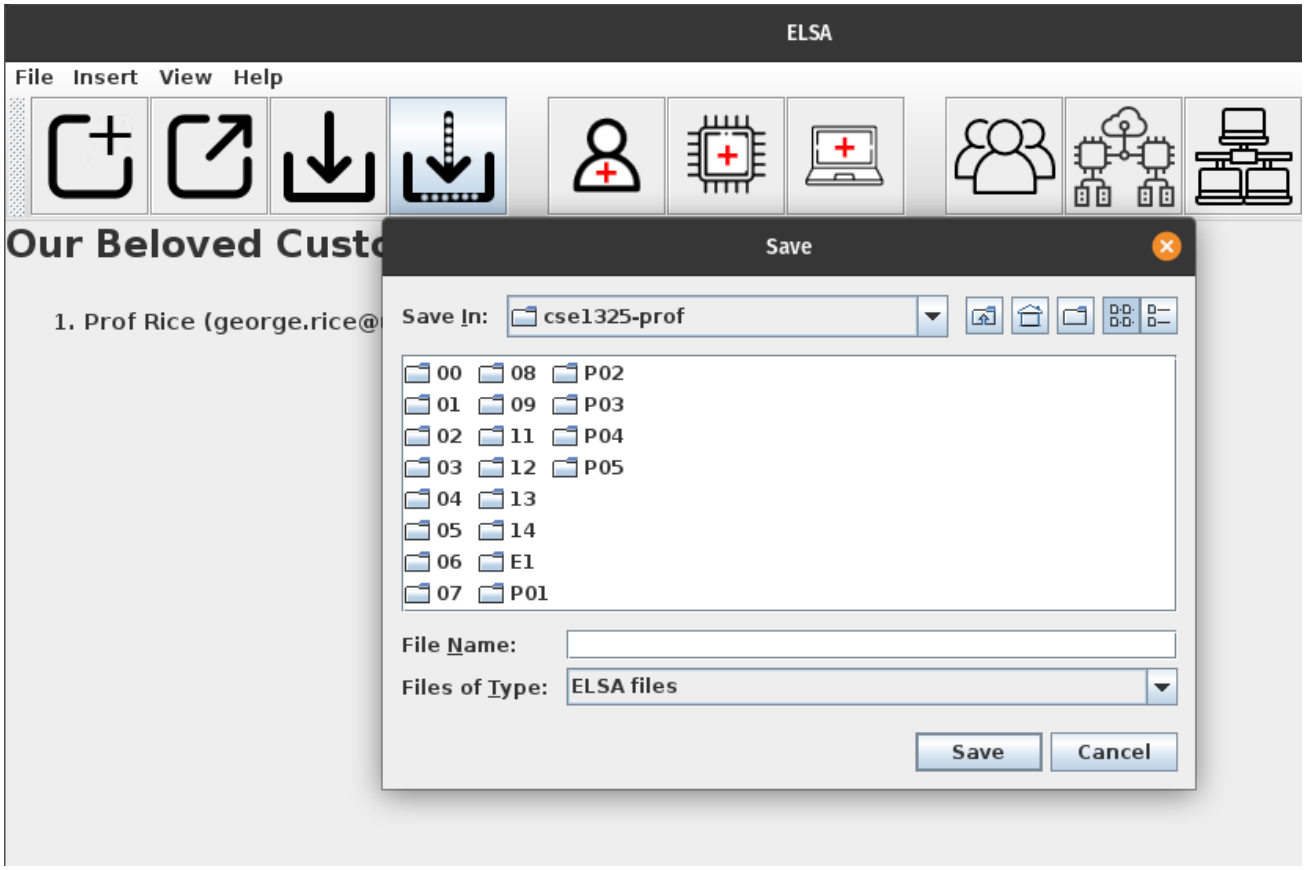
But the key method you absolutely *must* override is `public void paintComponent(Graphics graphics)`, which is called by the operating system whenever your drawing needs to be redrawn. In this method,

- Start by casting a clone (method `create`) of the Graphics parameter (the "crayon") to a Graphics2D variable, as we discussed in class. Cloning the "crayon" ensures you don't make any changes that affect other parts of the system.
- Set the color for what you are about to draw or print using Graphics2D's `setColor` method. Your drawing must contain at least 2 different line or text colors excluding Color.BLACK and Color.WHITE.
- Draw lines using Graphics2D's `drawLine` method. Use as many as you like. This is particularly effective in a loop with changing colors!
- Draw text somewhere on your canvas using Graphics2D's `drawString` method.
- Draw a BufferedImage (your logo, perhaps) somewhere on the Canvas using Graphics2D's `drawImage` method. Drawing this last allows it to overlay your sketch. Transparency (if any) should be automatically respected, enabling some very nice effects.

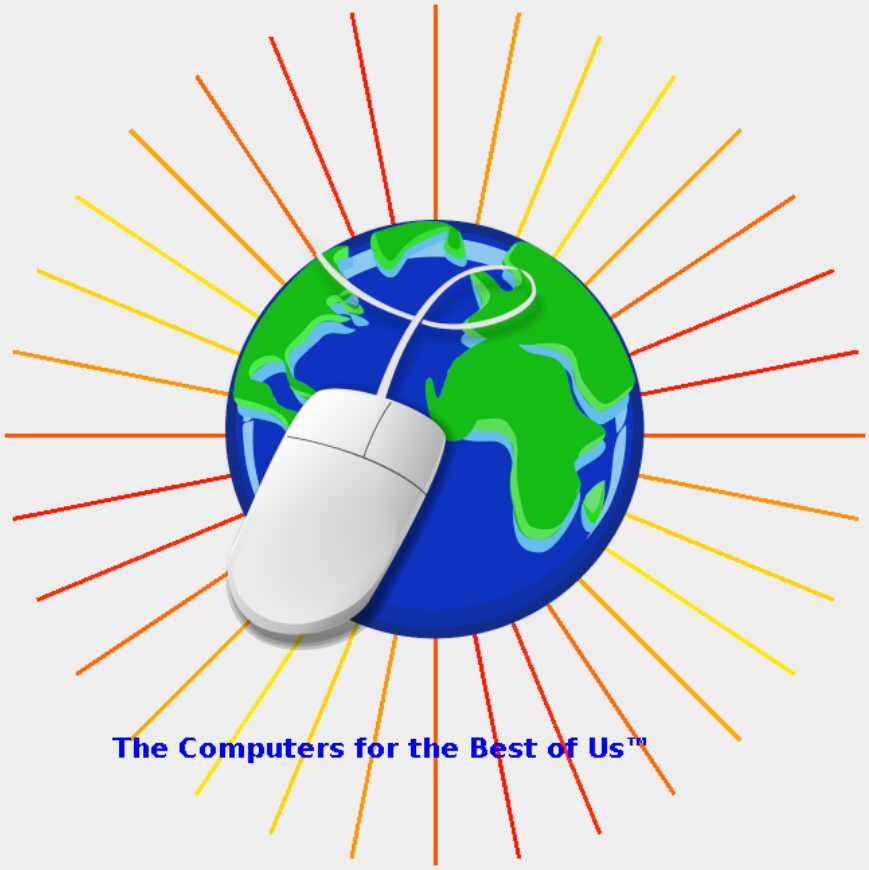
An instance of your Canvas class may be added to your Object[] array that you pass to JOptionPane's `showMessageDialog` call in your `MainWin.onAboutClick()` method, or included in your JDialog if you prefer.

Screenshots

IMPORTANT: Your application should NOT look exactly like this. **Make ELSA your own!** Be creative with your icons, your About dialog, and other visual effects.


ELSA
File Insert View Help
Our Beloved Custo
1. Prof Rice (george.rice@
Save
Save In:
cse1325-prof
00 08 P02
01 09 P03
02 11 P04
03 12 P05
04 13
05 14
06 E1
07 P01
File Name:
Files of Type:
ELSA files
Save
Cancel

ELSA



ELSA

Exceptional Laptops and Supercomputers Always
Version 0.3

Logo based on work by Clker-Free-Vector-Images per the Pixabay License
https://pixabay.com/vectors/internet-www-mouse-web-business-42583

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

New and open icon based on work by IconKanan per the Flaticon License
https://www.flaticon.com/free-icon/share_2901214

Save and Save As icons based on work by mavadee per the Flaticon License
https://www.flaticon.com/free-icon/download_3580085

OK