# Sprint 2 Guidance

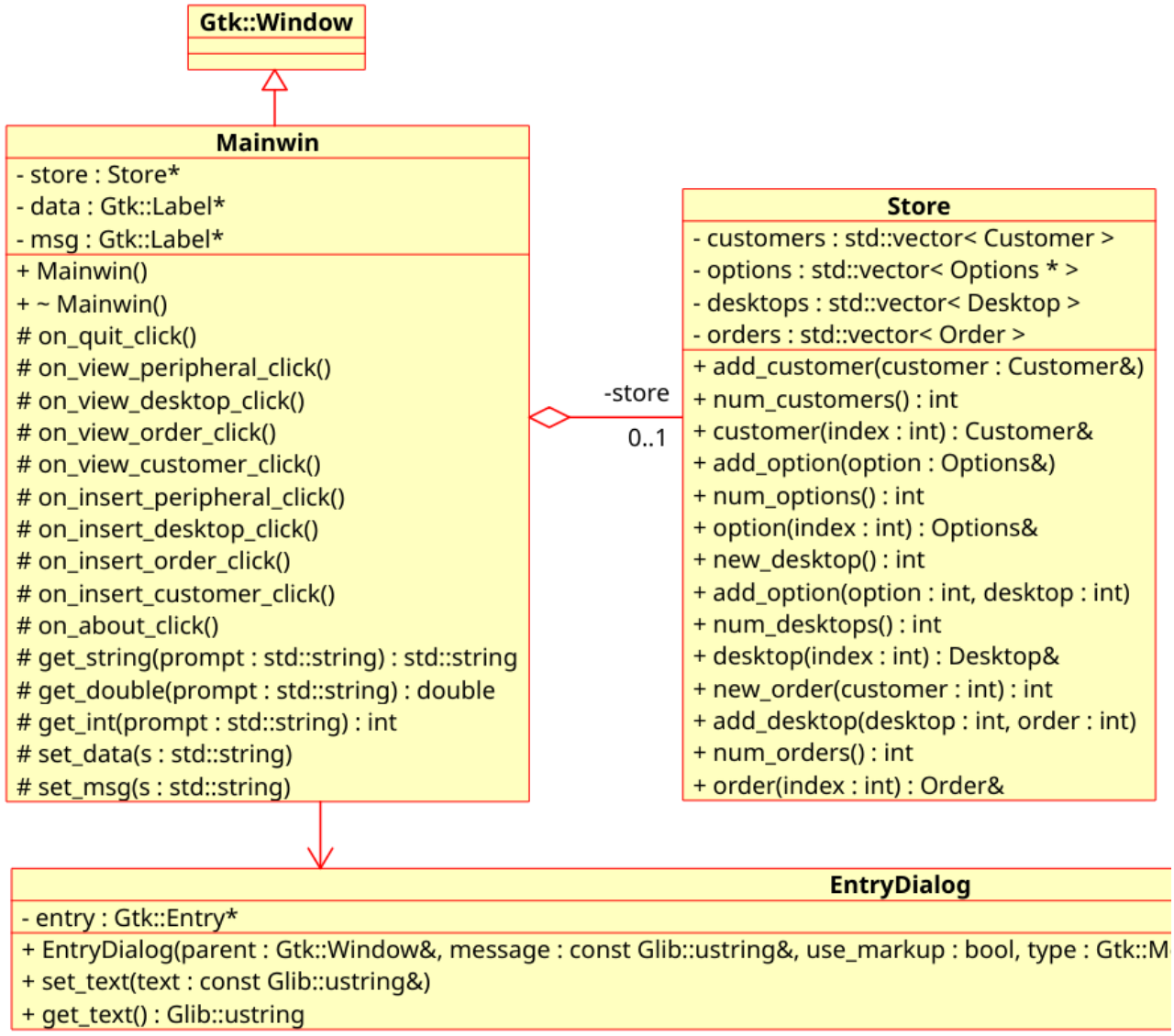
CSE 1325 – Spring 2020 – ELSA

**IMPORTANT:** Do NOT use full_credit, bonus, or extreme_bonus subdirectories for the final project. Keep all files to be graded in the cse1325/elsa directory on GitHub for the duration of the final project.

---

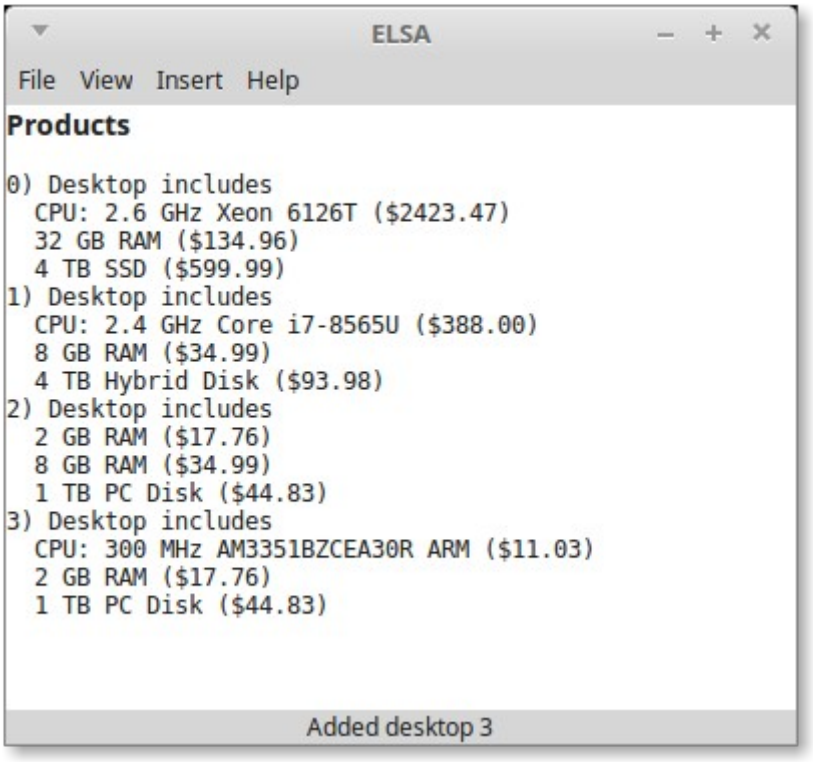
**For Sprint 2, you will replace the entire command line interface from Sprint 1 (principally the main function) with a main window and dialogs.**

This class diagram is for sprint 2 only (class EntryDialog is intentionally clipped at the edge):




While this sprint requires a bit more code in the suggested solution – 221 lines compared to 175 from Sprint 1 – much of it is repetitious and some of it can be baselined from Lecture 13's Nim game. As is often the case, it's not the number of lines of code but the new concepts that helpfully stretch the brain!

Your example main window might look something like this (NOT required). A brief description of each class follows.




# Class Overviews

## Store et. al.

The classes from Sprint 1 – Store, Order, Desktop, Options, and Customer – should need only slight changes if any. Only Store is shown in the class diagram, as it interfaces to the Mainwin class.

## Mainwin

This is the canonical main window that appears when most graphical programs are launched. I recommend that you *baseline*[1] the nim Mainwin class at **cse1325-prof/13/nim/mainwin**.*

The get_string, get_double, and get_int methods are optional, but will reduce the volume of code needed to get data from the user via dialogs. Similarly, the set_data and set_msg methods simplify putting data into the main window. Recommended!

You'll need a pointer to your store. You'll also need the pointers to the data area and message (status bar) in the main window so that you can modify their contents in the observers.

## EntryDialog

This optional class is a "quick and dirty" replacement for std::getline. You can find it at **cse1325-prof/14/02_entrydialog/entrydialog**.* You are free to modify it, although no changes should be required for this sprint.

## Main

This "class" (not shown in the diagram) is just the main() function, virtually identical to **cse1325-prof/13/nim/main.cpp**. The only change is to the 3rd parameter of the Gtk::Application::create static method call, with which you need to name your application.

---

1This just means you take the unmodified files from the original source and add / commit them to your current project's git repository, *then* modify them to fit your new requirements.

# Transitioning Mainwin to ELSA

Here's the order I followed in creating the main window.

1. **Create a Sprint Backlog.** Go to the Sprint 02 Backlog tab in the Scrum spreadsheet. As you read through the suggestions below, prior to coding, **create a list in the Description column of the tasks you'll need to accomplish** to successfully implement this sprint. Reorder and edit this list until you're comfortable with your plan.

   Then (and *only* then) **begin working through the list**, marking off tasks as they are completed by setting the Status column for that task to "Completed Day n" (where n is 1 to 7, the day you completed that task). It's perfectly OK to add and delete tasks as you go, and to reorder occasionally as you learn more about the task. No plan is perfect, but perfection is not possible without a plan.

   Developing the habit of starting with the end in mind, as Dr. Covey would recommend, will ensure that you waste less time spinning your wheels and spend more time on actually achieving your goals. By the way, *The 7 Habits of Highly Successful People* is a great book for computer and software engineers and scientists to read!

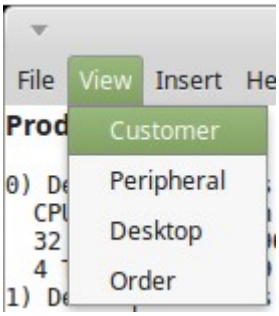
2. **Update mainwin.h.** Include "store.h" rather than "nim.h", of course. Keep on_quit_click() and on_about_click(), but replace the other callbacks (also called observers) with the on_view_* and on_insert_* observers shown in the class diagram. Also add the get_* and set_* utility methods if you'd like to use them. Replace the private data with pointers to your Store, and to the data and message (status bar) areas (you'll need them as attributes so that you can modify them in the observer methods).

   *Add to git, commit, and push!*

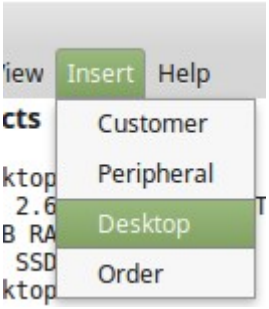
3. **Clean up mainwin.cpp** so that it compiles. Remove the observers you deleted from mainwin.h, and add empty methods { } for the ones you added. You can delete or comment out the tool bar, since it isn't required for this sprint. Also remove any references to Nim-specific data fields.

   *Ensure that Mainwin compiles and all code is on GitHub before continuing*, even though it won't run properly. Progress!

4. **Configure the main menu in Mainwin::Mainwin.** You should already have File > Quit and Help >About from your baseline. Add View and Insert menus, with Customers, Peripherals (or Options), Desktops, and Orders under each. This is a lot of "clone and own" work – copy, replace all of that method name with this method name, etc. Slog through it.




   Then, connect their Activate signals to each respective (empty) observer methods. When you build this time, your main window should *look* like the ELSA app, except that nothing happens when you click most menu items. It's a mock up of your application!




   *Ensure this compiles, runs and is on GitHub before continuing.*

5. **Implement the utility methods**, if you're using them. My get_string method instances an EntryDialog (don't forget to `#include "entrydialog.h"`), passing along the prompt string, then runs the dialog, then returns the text entered by the user via method get_text.

   My get_double and get_int methods use std::stod and std::stoi to convert the get_string return value to a double or int, respectively. I recommend that you catch any exceptions thrown, though. I quietly return -1.0 and -1 on an exception, since that's the "done" special value in the application code, which means when the user clicks Cancel in the entry dialog, it will be interpreted as "no more data". :-)

   The set_data and set_msg methods just call set_text (or set_markup, if you'd like to display Pango – hint, hint) for the respective labels. If you're using Gtk::Statusbar for the latter, call the push method instead.

   It's a *good idea* to test these before continuing, although in this case I was anxious enough to just plow ahead. Do put them into GitHub, though – as always!

6. **Implement Insert > Customer** (on_insert_customer_click). You may want to copy the code for "(C)reate a new Customer" from the CLI version to here and modify it. Basically, I get the customer's name using the get_string utility method, and if that' s not blank, also get their phone and email. Then instance a new Customer object and call Store::add_customer.

   As a nice added touch, also call on_view_customer_click (so the new customer is included in the data area), and set the status bar to something like "Added customer [name]".

   *Ensure that this compiles and runs and is on GitHub before continuing.*

7. **Implement View > Customer** (on_view_customer_click). Again, you may want to copy the code for "List (c)ustomers" from the CLI version and modify it. This code streams out the text to std::cout – but we need to capture that text for our main window data area. What to do? Stream to a string – a *string stream*.

First, **`#include <sstream>`** at the top of mainwin.cpp.

Next, instance an output stream string with **`std::ostringstream oss`**. You may now **replace std::cout with oss** in the rest of your copied code – they work identically!

Finally, set your data area (set_data, if you're using the utility methods) to **`oss.str()`**, which returns all of the text we streamed to oss. For good measure, clear the status bar. Done!

*Ensure that this compiles and runs and is on GitHub before continuing.* At this point, you should be able to create a new customer (Insert > Customer) and view those customers (View > Customers) entirely within the graphical user interface.

8. **Repeat the two previous steps for Peripherals** (or Options), then **Desktops**, then **Orders**. They are similar but with minor twists to give you practice. After each step, ensure the code builds and runs, then commit to GitHub before continuing to the next step.
9. Finally, **update Help > About** (on_about_click) to reflect the ELSA brand. You may want to create a logo for the company if you have time. It makes your program look more polished.

And now you've created your first user interface in gtkmm! I recommend that you **`git tag sprint2`**, then **`git push --tags`**.

# What Is To Come

In **Sprint 3**, we will load and save our data to a file. I'll show you a simple approach that enables you to focus independently on each class, programming it to save and load its own data to and from a stream, and then the file format will fall out naturally out of the data to be saved.

**Exam #2** then distracts us from our project for a week.

In **Sprint 4** we'll replace one of the sequence of dialogs in this sprint's implementation with a single custom dialog. Instead of a dialog asking for the customer's name, then another asking for their phone number, then yet a third asking for the email, you'll present a single dialog with 3 labeled Gtk::Entry widgets, and Add button, and a Cancel button. Better!

In addition, we'll collect an additional attribute, the number of gigabytes, for each RAM peripheral. This means we'll need to invoke the operator<< function *polymorphically*, which is a bit of a trick since polymorphism only works with classes. :-D Not to worry, we'll walk you through it all.

**Bonus Features:** To earn additional credit, you may implement additional features beyond those required. For example, adding other derived classes for disks or CPUs, replacing the other sequences of dialogs with custom dialogs, making the data area in the main window scrollable,

Our project is complete at this point, though we'll hopefully have time for a few students to present their projects to the class (for extra credit, of course). Our last assignment will be a **stand alone multi-threaded app**, and we're on **Exam #3** and wrapping it all up with a bow on top!