CS106X
Cynthia Lee Autumn 2015

Assignment 8: Stanford 1-2-3 GUI Extension Primer

Handout written by Ilan Goodman

What is this extension?

You may have noticed over the course of programming your Stanford 1-2-3 that it's fun to play around with the spreadsheet using text commands in the console, but it would be much more convenient if you could interact directly with the spreadsheet GUI and type directly in there. If you try to do so right now, the changes will persist in the GUI, but they do not affect the backend at all. The table interprets all strings literally and it does not update our internal representation of the data. Essentially it's a one-way pipe of information: the console commands will interact with our internal data state and will become reflected on the table GUI, but any information fed into the GUI stays at that level. We'd like to change this.

Cool! How do I do this?

Depending on how fancy you'd like to make it and how you designed your program to begin with, you probably won't have to change too many additional files. You will definitely need to update `sscontroller.cpp`, may want to check out `ssview.h/cpp`, and could need to tweak a couple other files.

Let's start by adding some interactors to the screen! There's a class called `GButton` (take a look at the online documentation to familiarize yourself) in the Stanford libraries. If you go into `ssview.h/cpp`, you can add `GButtons` to the window using syntax similar to:

```
GButton mybutton("My Action Command and Label");
addToRegion(&myButton, "SOUTH");
// This button takes space, so also update the GWindow dimensions
```

Be sure to think about whether you need the `GButtons` to be member variables or if they can exist locally. Notice that the constructor for a `GButton` takes a `string` that functions both as a label and as an action command. An action command serves as an identifier to know which button was clicked on when we get an event notification in our event loop. Speaking of which …

Take a look at the `interpretCommands` function in `sscontroller.cpp`. The core of this function is a loop that continually waits for the user to give a command and then acts based on that command. You can do a similar thing for events, known as an event loop. Your event loop will probably want to handle events from the table, the other interactors you set up, and the window in case someone accidentally closes it. You will have to `#include "gevents.h"` as well and your code will probably take the following structure:

```
while (true) {
    GEvent event =
        waitForEvent(ACTION_EVENT | TABLE_EVENT | WINDOW_EVENT);
    if (event.getEventClass() == ACTION_EVENT) {
        // Respond to button presses
        GActionEvent actionEvent(event);
        if (actionEvent.getActionCommand() == "cmd1") {
            // Handle cmd1
        } else if (...) {
            // You can handle more command types
        }
    } else if (event.getEventClass() == WINDOW_EVENT) {
        // Respond to closing of window by stopping event loop
        if (event.getEventType() == WINDOW_CLOSED) {
            break;
        }
    } else if (event.getEventClass() == TABLE_EVENT) {
        // Respond to table cell updates
        GTableEvent tableEvent(event);
        // Handle tableEvent
    }
}
```

When you write this, think about what buttons need to be added, what should happen when a cell is updated in the table, what events are disallowed (you should not allow a user to update the row or column labels, for example, and you still cannot let them create a dependency loop), and what tricky cases there are (how do you handle a string literal vs. a number vs. a formula or cell reference?). You also need to think carefully about how you want the console and GUI to interact: you can get rid of the console entirely, you can listen for both events and console input, or you can give the user a choice for some sort of compromise. The documentation for `GTable` may be of use (please look through the documentation for all these classes when you are unsure what you can use). Note that you do not need to get every last detail perfect to receive extra credit. Focus on making the table GUI interaction smooth, and from there you can add more nuance and other interactors.

If you decomposed your code well enough in the original version of the assignment, much of the event handling should be repurposing old code and it shouldn't be too big of an issue to plug in. Be sure to unify code wherever possible and don't forget to test your program thoroughly as you work your way through it. Please also remember to use constants where appropriate, and absolutely document everything you do (we want to understand the choices you made).

Anything else I need to know?

This is your extension, so do with it whatever you think will make the coolest spreadsheet possible. Do you want to get rid of the console entirely and just interact with the table and additional interactors? Do you want an option up front for the user to choose console or interactors? Do you want an option every command? Do you want to listen for both text and events at the same time? Do you only want to handle SETting in the GUI or do you want to cover all types of commands? Do you want to add something not detailed here? All these design choices are up to you, but please be sure to document everything you do clearly and carefully.

As always, please submit one set of files with the basic functionality and no extensions separately from the version with extensions to ensure that the extensions don't mess up the grading system in place. Also remember that extensions are graded for style as well, so be sure to write clean code both for the basic functionality and the extensions you implement.

Good luck and have fun!