Final Project Proposal

Foundations of Computer Science, TJHSST 2023-2024

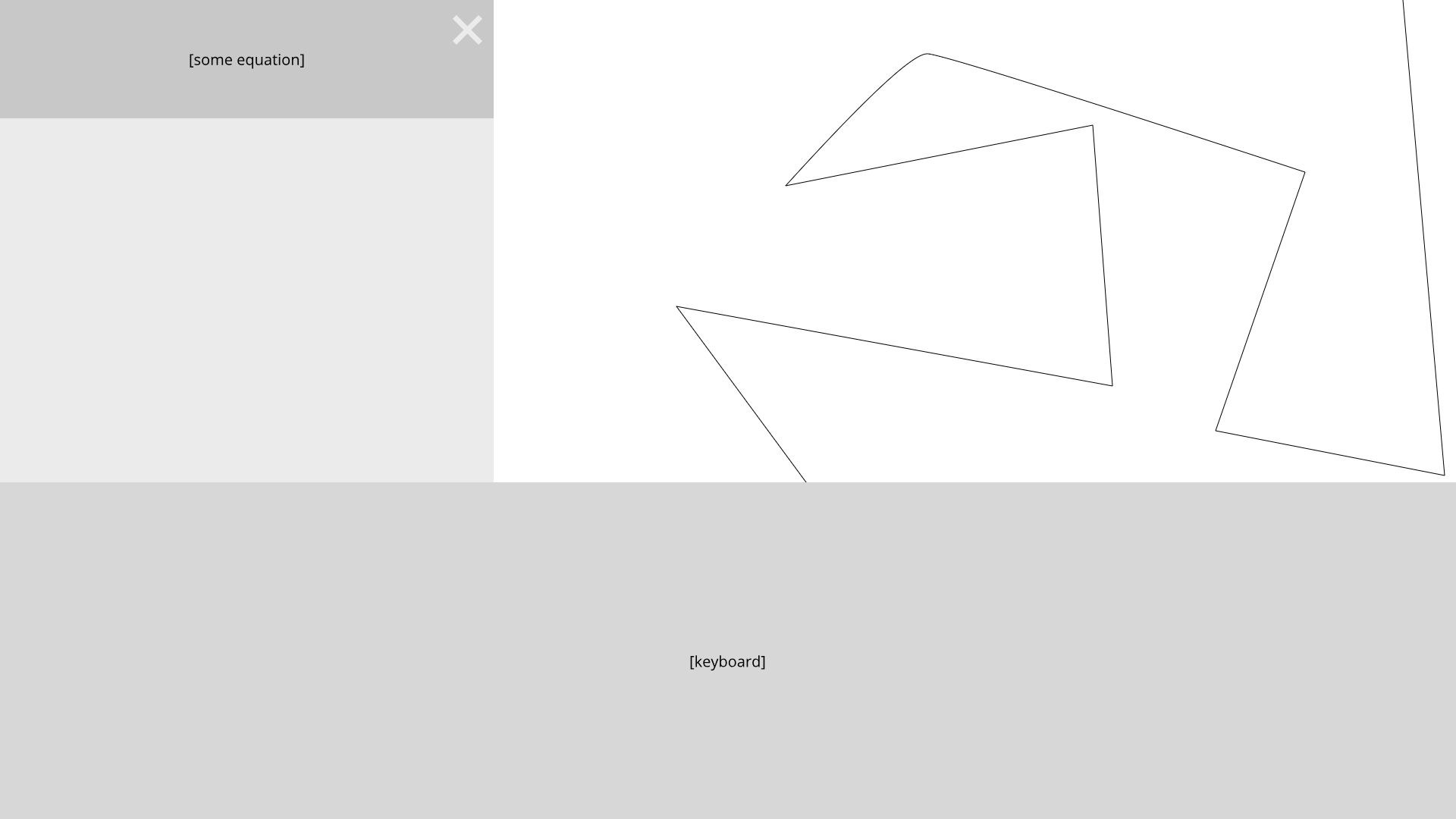
Write the names of everyone in your group here:

Eshan Iyer

Samuel Stankiewicz

# 1) What will the user SEE when they run your code?

Do sketches of each separate screen that the user will see when running your project. If it’s a game, what will gameplay look like? Is there a loading screen? An options select screen? A pause screen? These can be very, very sketchy; feel free to use MS Paint or the equivalent, or draw by hand.



# 2) What can the user DO when they run your code?

Describe below all of the interactivity you’re planning. If it’s a game, what will gameplay feel like? What keys or mouse inputs does the user have, and what do they do? What can the user click? What GUI elements are there, and what do they do?

1. **Non-Graphing Capabilities**
   1. This section functions as a bog-standard calculator
   2. Basically, the user will be able to add, subtract, divide and multiply floating point numbers
   3. The inputs can either be via the keyboard or the mouse
   4. Integrated into the graphing panel sort of like Desmos
2. **Graphing Capabilities**
   1. This section allows a user to graph functions
   2. Will take up the majority of the screen

# 3) Requirements Checklist

Fill in the checklist spaces below for the requirements you plan to complete. If you’re substituting a harder requirement, make a note of it. You can’t really write too much here; I’m expecting you to write a lot and be very clear!

ACCELERATED CREDIT (100%)

| valid use of an abstract class you write (explain how you’ll use this to avoid writing repeated code over several different classes) OR one interface you write yourself (explain how this will enable a similar functionality across many different objects) |  |
| --- | --- |
| use of mouse input (not including GUI elements like JButtons) | Mouse input can be used to pan and move the graph |
| a valid use of something in your code that we never taught you (you should ask your teacher about this and make sure we agree it’s complex enough in advance) |  |
| a program of extreme complexity or artistic merit  (you should ask your teacher about this and make sure we agree it’s complex enough in advance) |  |
| subtle, comprehensive use of object oriented programming (such as a 4-level object hierarchy or use of multiple appropriate interfaces, etc) |  |

RED CREDIT (94%)

| design at least 10 of your own classes total, including the 5 for the BLUE requirement (including the abstract class if applicable; not including driver or Listeners) | * User input class * Keyboard class * Graphing class (physically plotting data) * Computational class (add, subtract, etc.) * Advanced functions class (exponents, logs, etc.) * Panel manipulation class (ie, panning, zooming, etc, possible extension of ui or graphing class) * Assorted functions class (anything we’ve forgotten somewhere else) * UI class (rest of the screen excluding graphing and keyboard) * Animation class (extension of ui class) * User flow class (moving around ui, ui controller, etc.) |
| --- | --- |
| objects that interact (ie, create an object that has a method taking another object you’ve written as an argument, or uses one as a field; graphics panels/subpanels don’t count) | The graphing class will take the user input class as an input and use this to parse incoming data. |
| use file I/O (save and/or load some info from a file) - don’t forget to complete the file I/O additional assignment first! | XXX -- Swapped for mouse input |
| use panel swapping - don’t forget to complete the panel swap additional assignment first! | When the function has been graphed, the initial blank graphing-section will be swapped with a panel where the graph has been plotted. |

BLUE CREDIT (87%)

| use 2d arrays | We’ll keep all the user input buttons in a 2d array including the keyboard |
| --- | --- |
| valid use of a built-in Java interface or our Animatable interface | The Animatable interface will be employed in order to graph the lines and for transitioning. |
| at least one complex algorithm (nested for loops or recursion) | A nested for loop or recursion could be used to plot lines on our graph. |
| design at least 5 of your own classes total, including the 2 for the GREEN requirement (not including the driver or Listeners) | * User input class * Keyboard class * Graphing class (physically plotting data) * Computational class (add, subtract, etc.) * Advanced functions class (exponents, logs, etc.) |
| use animation (via a Timer & actionListener) | We can provide minor animations such as fading and transitioning for user input and other user-interface tasks. |

GREEN CREDIT (80%)

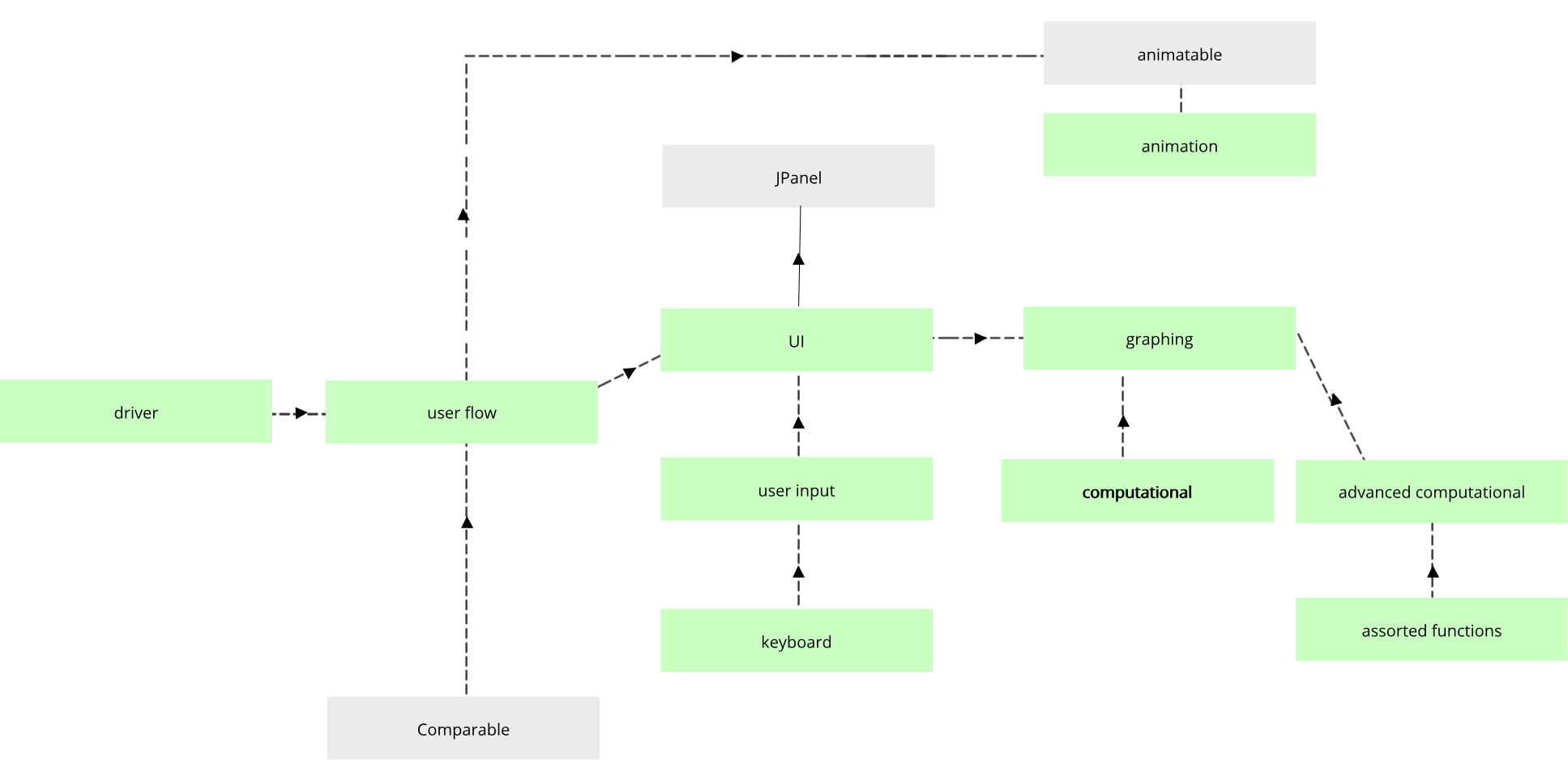
| code in Java | We’ll code in Java |
| --- | --- |
| use loops – for and/or while | Presumably this will require some kind of for loop somewhere, we’re unsure where though. |
| use strings | The non-graphing section of the calculator will display the calculated value as a string |
| use arrays | Arrays will be used in our application at some point |
| design at least 2 of your own classes (not including the driver or any Listeners you use), including accessor methods, modifier methods, and two different constructors | We’ll have separate classes for the graphing panel and the user input panel, as well as for the keyboard and external logic. |
| use inheritance (one of your classes must extend another class) | Operation and equals buttons will extend the JButton class in order to create stylized buttons |
| use GUI elements (JPanel, JButton, etc) | Our app is a GUI application, requiring JPanels and JButtons |
| use graphics (use of images or drawing shapes) | Graphics will be used in order to plot the lines |

# 4) UML Diagram

Draw a UML diagram of your planned classes. We know this plan may change, but it’s better to have a plan and modify it than it is to have no plan and get stuck! [www.draw.io](http://www.draw.io) is a useful tool to draw your diagram but feel free to use whatever you want!

Include:

* Green boxes for classes, blue boxes for interfaces you write
* White or grey boxes for any built-in Java classes or classes provided by us that you plan to use (ie, Animatable)
* Solid black arrows to represent inheritance (“isa” relationship)
* Dotted blue arrows to represent implementation of an interface (another kind of “isa” relationship)
* Dotted black arrows to represent when one object constructs/owns another (“hasa” relationship)



# 5) Additional Requirements for Unusual Situations

If you’re a pair of partners in the same class period, or someone working alone, **you’re done**.

If you’re a pair of partners working in **different** class periods, or a group of **three**, in the space below divide up the work in advance. How are you going to ensure that everyone has something they can work on individually?

**N/A; Pair of partners working in the same class period.**