Some thoughts about the logical layout and flow of the program follow. These are just some ideas that will surely change as more accurate ideas form.

==

How to: Load raw data file

- Have a class called RawDataLine that has a data member for each point of data in a line of the file.
- Methods:
  - Read an instance of itself in from the file.
- Constructor accepts an open stream to the file being read.

Questions:

- Who has an instance of this class?
  - Calculator has a list of instances of this class.
- What kind of stream should be used?
  - ?

==

How to: Write calculated data to file

- Have a class called OrderedPair that has two data members, one for time and one for either acceleration, velocity, or displacement based on which calculated value it represents.
- Methods to:
  - Write itself to a file with an open stream passed to it when called.
  - Read itself from a file with an open stream passed to it when called.
- Constructor sets initial values.

Questions:

- Who has an instance of this class?
  - Calculator has a few instances of this class that it tells to write to the appropriate graph data file. To elaborate, one instance for time versus acceleration, one for time versus velocity, and one for time versus displacement.
- What kind of streams should be used?
  - ?

==

How to: Calculate the data

- Have a class called Calculator that tells several RawDataLine instances to read themselves in from the raw data file and then, using the appropriate data members, to calculate required results such as velocity and displacement at a given time in a given axis. These resulting data points are turned into instances of OrderedPair that will be read in and graphed later, but for now are told to write themselves to file.
- Methods to:
  - Do calculation on RawDataLines currently stored in a list to create instances of OrderedPair that then are told to write themselves to their respective graph data files..
- Constructor accepts:
  - Raw file path to open a stream on.
- This class opens the streams on every file being read and written to, one for reading the Raw data file and many for writing the pairs of graph data to files.

Questions:

- Who owns this class?
  - Likely when the drop down menu item to 'Load Raw Flight File' is clicked, or a file is drug onto the screen that is a Raw Flight File, this class is instantiated and passed the file path so it can open it and begin reading.
  - How will we know what files are Raw and which are graph data files? Signature at the beginning of the graph data files?
- How are the graph data files named and placed?
  - Should the user pick, or should their be a hard coded location for each flight?

==

How to: Read a Graph data file

- Have a class owned by the new JTabbedPanel that opens a stream on every graph data file in this flights directory and starts having instances of OrderedPair read themselves in.
- Methods to:
  - Open a stream on a graph data file and test for signature.
  - Create OrderedPair instances to tell them to read themselves in.
- Constructor accepts
  - Flight directory path?

Questions:

- What kind of streams should be used?

==

Given acceleration and time

A (t) = a
V (t) = Integral (A(t)) dt = a * t + v_0
S (t) = integral (V(t)) dt = (a * t ^ 2) / 2 + v_0 * t + s_0

v_0 is the previous velocity in this case
s_0 is the previous displacement in this case

Riemann Sum Approach

ΔXi = Xi - X(i-1)

Xi is a time, X(i-1) is the time before Xi

Ci = the acceleration value in a given axis, the acceleration at time ΔXi

Area under curve = $\sum_{i=1}^{n} f(Ci) \, * \, \Delta Xi$

So, to find Velocity in the x axis…
Given

Time	Accel in x axis
t1	a1
t2	a2
t3	a3

Right hand rule

V1 = f (a2) * (t2 - t1)
V2 = f (a3) * (t3 - t2)

f (ai) = ai * ti + v(i-1)

V1 = (a2 * t2 + 0) * (t2 - t1) , since initial velocity will be assumed 0 for the rocket
V2 = (a3 * t3 + V1) * (t3 - t2)
...and so on…

Left hand rule

V1 = f (a1) * (t2 - t1)

Trapizoid Rule

V1 = f ((a1 + a2) / 2) * (t2-t1)

This is what will be used in the software, except that closer to ten acceleration points will be averaged to smooth out the outlying data points.

Final x axis equations:

$V_i = (((a_i + a_{(i+1)}) / 2) * t_i + V_{i-1}) * (t_{(i+1)} - t_i)$
$S_i = (((a_i + a_{(i+1)}) / 2) * t^2 / 2 + V_{i-1} * t_i + S_{(i-1)})$

==

The spin of the rocket must be accounted for too. Each acceleration point must be calculated based on the current spin of the rocket using trig. Need to come up with that math still.

Have the think about correcting in the z axis before correcting in the other two, but need to consult with someone to gain a better understanding of this problem and potential solutions.