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.

Project Flow: Loading a raw data file.

Program opened → Begin Loading Raw Datafile → Begin Calculating Data → Begin Saving Calculations → Finish Data Calculation and close all files

==========

How to: Load raw data file

- Have a class that holds a stream open on the raw data file.
- Methods to:
    - Read a single line from the file and return it.
    - Close the stream on the file.
- Constructor accepts a file path to open the stream.

Questions:

- Who has an instance of this class?
    - Maybe the class calculating the data has an instance of this class so it has the ability to call the appropriate methods and accept the returning line.
- What kind of stream should be used?
    -

==========

How to: Write calculated data to file

- Have a class that holds a stream to write to a file.
- Methods to:
    - Accept and write a line to the file in the correct format to graph it.
    - Close the stream on the file.
- Constructor accepts a file path to open the stream on.

Questions:

- Who has an instance of this class?
    - Maybe the class calculating the data has an instance of this class so it has the ability to call the appropriate methods and send the calculated line to be written.
- What kind of stream should be used?
    -

==========

How to: Calculate the data

- Class that requests several lines of the raw file to begin parsing the lines and using the appropriate data points to calculate required results such as velocity and speed at a given time. These resulting data points are ordered into the pairs they will be graphed in and sent to a class that is meant to write them to file.
- Methods to:
    - Request another line of data (this is just calling a method of the class reading the raw file)
    - Do calculation on currently stored points, stored in some type of list?
    - Send resulting data point pairs in graphable format to class writing to files. (This is just calling a method of the class writing to a file)
- Constructor accepts:
    - Raw file path so reading class can open a stream on it
- Has instances of the read and write classes. Shouldn't just be one big class right? This class could just have the methods of the classes it has instances of, but that is probably not OOP enough.

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 the file reading class 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 to: Read a Graph data file

- Have a class owned by the current JTabbedPanel that opens a stream on every graphical data file in a certain flights directory and starts populating the data structure the graph wants. (Series?)
- Methods to:
    - Open a graph data file and test for signature of said file to ensure it is actually a graph data file.
    - Read a line from the file and return it
- 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 speed 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.

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.