Executive Summary

Our project, Weight and Balance in Aircraft, was intended to provide a way for a pilot to determine if his aircraft is balanced and within its weight limits quickly and easily. Our program will be written in the Java programming language. We chose Java because it was what we were learning in our Computer Science class and we thought it would provide the easiest way to make our program work. We chose this subject because one of our group members was studying for his pilot's license and noticed how long it took to calculate weight and balance. For a pilot to calculate the weight and balance of an aircraft by hand, he must find all the arms and weights of the components of his aircraft and then, using those values, find each components moment, add the moments with the aircrafts center of gravity and then plot the total moment on a graph. As you can see, this can be a time consuming process. Our program was originally designed as an all purpose weight and balance calculator that could be used in any fixed wing aircraft, but we soon discovered that all the values for fixed wing aircraft were hard to find so we focused our program on a single airplane, a Cessna 172N. We found all the formulas needed from a Cessna 172N Pilots Manual. We also needed formulas to convert the total moment and weight into screen position to graph them on our Applet window. We came up with our own formula to fit our specific needs. Our program is a hybrid of a Java JFrame and an Applet's AWT window. We used a JFrame and JOptionPanes to get the values from the user, and the AWT window to display the graph and values. The program displays the data inside an envelope taken from a Cessna 172N Manual.

Problem

The main problem that we had to tackle was how to make the program easy enough for the average pilot to use, but still have it be faster than doing the calculations by hand. We employed an easy to understand GUI (Graphical User Interface) and an end graph to make the input and evaluation of data as easy as possible. Most of the weight and moment formulas were relatively easy to find.

Terminology

Datum – A set Reference point (usually the firewall) that all distance measurements are made from.

Arm – Distance from a component of the aircraft to the datum (inches)

Moment – Arm * Weight (pound per inch)

Firewall – where the engine is attached to the airplane

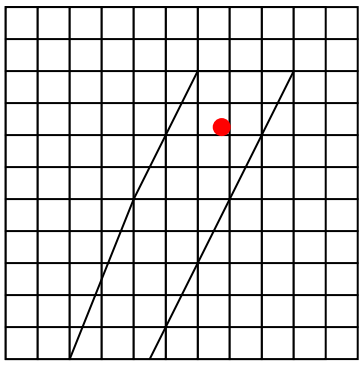
AWT – Abstract Window Toolkit (used to draw graphics in Java)

Methods

We used input from real pilots to determine if our program was easily useable. Because the early builds of our program were text-based, the process of inputting data quickly became a confusing and time-consuming process. The later versions of our program adopted Pop–up windows and a simpler input interface. These versions were easier to use and interpret, but it still left the pilot to determine if the given center of gravity was acceptable for the aircrafts weight. The current build of our program has one small but significant change over the last version, it displays the final data in a graph with the envelope of weight and balance for a Cessna 172N. If the point where moment and weight intersect is within the envelope, then the aircraft is balanced. The latest version received a bill of approval from our mentor, Lieutenant Colonel Charles Hainline.

Results

The results of our project were very favorable. Our program exceeded almost all of our expectations. The one area that it did not excel in was that it only simulated the weight and balance of one aircraft, the Cessna 172N. The following picture is a sample of how the program works in action.



The graph shows that the center of gravity lies inside the balance envelope and the airplane is therefore balanced correctly. In order to make the program simpler and easier to use, we removed some of the lighter components from our calculations because they did not provide a big enough change to really affect our final values.

Conclusions

We came to the conclusion that our program drastically reduced the time it takes to determine if an aircraft is balanced. We also concluded that Java is a very good programming language to use when making small, powerful applications. We think that our program will allow pilots to spend less time calculating weight and balance and more time in the air.

Source Code

```
import javax.swing.*;//imports a java library
import java.awt.*;//imports java core package

public class Balancer extends JFrame{

//Data Declaration

String emptycg;
String emptyweight;
String fuelweight , fuelarm;
String pilotsweight , pilotsarm;
```

```
String passengersarm , passengersweight;
String baggageweight , baggagearm;
int totalmoment;
 int totalweight;
int emptycg1;
int emptyweight1;
int fuelweight1 , fuelarm1;
int pilotsweight1 , pilotsarm1;
int passengersarm1 , passengersweight1;
 int baggageweight1 , baggagearm1;
//end of data declaration
public Balancer()//sets up GUI
{
super("Weight & Balance");//sets Title Bar message

emptyweight = JOptionPane.showInputDialog("Enter the empty weight of your
aircraft");//gets value from user
emptycg = JOptionPane.showInputDialog("Enter the arm of your airplanes center of
gravity");//gets value from user
fuelweight = JOptionPane.showInputDialog("Enter the amount of fuel (in pounds) in you
airplane");//gets value from user
fuelarm = JOptionPane.showInputDialog("Enter the arm of the fuel tanks from the
datum");//gets value from user
pilotsweight = JOptionPane.showInputDialog("Enter the weight of the pilot (and front
passenger)");//gets value from user
pilotsarm = JOptionPane.showInputDialog("Enter the arm of the pilot and front
passenger)");//gets value from user
passengersarm = JOptionPane.showInputDialog("Enter the arm of the rear passengers
from the datum");//gets value from user
passengersweight = JOptionPane.showInputDialog("Enter the weight of the rear
passengers");//gets value from user
baggageweight = JOptionPane.showInputDialog("Enter the weight of the
baggage");//gets value from user
baggagearm = JOptionPane.showInputDialog("Enter the arm of the baggage from the
datum");//gets value from user

emptyweight1 = Integer.parseInt(emptyweight);//converts string to integer
emptycg1 = Integer.parseInt(emptycg);//converts string to integer
fuelweight1 = Integer.parseInt(fuelweight);//converts string to integer
fuelarm1 = Integer.parseInt(fuelarm);//converts string to integer
pilotsweight1 = Integer.parseInt(pilotsweight);//converts string to integer
pilotsarm1 = Integer.parseInt(pilotsarm);//converts string to integer
passengersarm1 = Integer.parseInt(passengersarm);//converts string to integer
passengersweight1= Integer.parseInt(passengersweight);//converts string to integer
baggageweight1 = Integer.parseInt(baggageweight);//converts string to integer
baggagearm1 = Integer.parseInt(baggagearm);//converts string to integer
```

```
totalmoment = (emptyweight1 * emptycg1) + (fuelweight1 * fuelarm1) + (pilotsweight1
* pilotsarm1) + (passengersweight1 * passengersarm1) + (baggageweight1 *
baggagearm1);//gets total moment
totalweight = emptyweight1 + fuelweight1 + pilotsweight1 + passengersweight1 +
baggageweight1;//gets total weight
totalmoment = totalmoment / 1000;//divides totalmoment to a more managable number


JOptionPane.showMessageDialog(null , " " + totalmoment + " pounds per inch / 1000   "
+ totalweight + " pounds");//displays total moment and total weight


setSize(150 , 245);//sets Window size to 150 across 245 down
setVisible(true);//sets GUI to be visible
}
public void paint(Graphics g)//graphics method
{

g.drawRect(0 + 5, 0  + 25, 10 , 240);//draws graph lines
g.drawRect(10 + 5, 0 + 25, 10 , 240);//draws graph lines
g.drawRect(20+ 5 , 0 + 25, 10 , 240);//draws graph lines
g.drawRect(30+ 5 , 0 + 25, 10 , 240);//draws graph lines
g.drawRect(40+ 5 , 0 + 25, 10 , 240);//draws graph lines
g.drawRect(50+ 5 , 0 + 25, 10 , 240);//draws graph lines
g.drawRect(60+ 5 , 0 + 25, 10 , 240);//draws graph lines
g.drawRect(70+ 5 , 0 + 25, 20 , 240);//draws graph lines
g.drawRect(80+ 5 , 0 + 25, 20 , 240);//draws graph lines
g.drawRect(90+ 5 , 0 + 25, 20 , 240);//draws graph lines
g.drawRect(1005 + 5, 0 + 25, 20 , 240);//draws graph lines
g.drawRect(110 + 5, 0+ 25 , 20 , 240);//draws graph lines
g.drawRect(120 + 5, 0+ 25 , 20 , 240);//draws graph lines
g.drawRect(130 + 5, 0+ 25 , 20 , 240);//draws graph lines
g.drawRect(140 + 5, 0+ 25 , 20 , 240);//draws graph lines
g.drawRect(0 + 5, 230+ 25 , 160 , 10);//draws graph lines
g.drawRect(0 + 5, 220+ 25 , 160 , 10);//draws graph lines
g.drawRect(0 + 5, 210+ 25 , 160 , 10);//draws graph lines
g.drawRect(0 + 5, 200+ 25 , 160 , 10);//draws graph lines
g.drawRect(0 + 5, 190+ 25 , 160 , 10);//draws graph lines
g.drawRect(0 + 5, 180+ 25 , 160 , 10);//draws graph lines
g.drawRect(0 + 5, 170+ 25 , 160 , 10);//draws graph lines
g.drawRect(0 + 5, 160+ 25 , 160 , 10);//draws graph lines
g.drawRect(0 + 5, 150+ 25 , 160 , 10);//draws graph lines
g.drawRect(0 + 5, 140+ 25 , 160 , 10);//draws graph lines
g.drawRect(0 + 5, 130+ 25 , 160 , 10);//draws graph lines
```

```
g.drawRect(0 + 5, 120+ 25 , 160 , 10);//draws graph lines
g.drawRect(0 + 5, 110 + 25, 160 , 10);//draws graph lines
g.drawRect(0 + 5, 100 + 25, 160 , 10);//draws graph lines
g.drawRect(0 + 5, 90 + 25, 160 , 10);//draws graph lines
g.drawRect(0 + 5, 80+ 25 , 160 , 10);//draws graph lines
g.drawRect(0 + 5, 70 + 25, 160 , 10);//draws graph lines
g.drawRect(0 + 5, 60 + 25, 160 , 10);//draws graph lines
g.drawRect(0 + 5, 50 + 25, 160 , 10);//draws graph lines
g.drawRect(0 + 5, 40 + 25, 160 , 10);//draws graph lines
g.drawRect(0 + 5, 30 + 25, 160 , 10);//draws graph lines
g.drawRect(0 + 5, 20 + 25, 160 , 10);//draws graph lines
g.drawRect(0 + 5, 10 + 25, 160 , 10);//draws graph lines
g.drawRect(0 + 5, 0 + 25, 160 , 10);//draws graph lines
g.fillRect(0 , 145 , 200 , 200);//draws black box for values
g.setColor(Color.red);//sets draw Color to red
g.drawLine(57 , 145, 68 , 240 - 165);//draws envelope
g.drawLine(68 , 240 - 165 , 88 , 45);//draws envelope
g.drawLine(88 , 45 , 109 , 45);//draws envelope
g.drawLine(109,45,80 , 145);//draws envelope
g.setColor(Color.blue);//sets draw color to blue
g.fillOval(totalmoment  - 2 , 240 - (totalweight / 10 - 35) , 4 , 4);//draws point
g.setColor(Color.white);//set draw color to white
g.drawString("Center of Gravity : " + totalmoment ,10 , 160 );//writes center of gravity
g.drawString("Total Weight : " + totalweight + " lbs" , 10 , 190);//writes total weight
}

public static void main (String args[])//Executes program
{
Balancer application = new Balancer();//instantiates new Balancer object
}
}
```

Formulas

The following formulas were obtained from a Cessna 172N Information Manual.

*Moment = Weight * Arm*
Total Moment = (all moments added together) + (aircraft center of gravity when empty)

The following formulas were designed by our group to fit our specific needs of converting values to x and y positions.

y-position = (total weight ⁄10 – 35)

x-position = total moment – 5

We got these formulas through trial and error after many days of thinking through the problem.

Achievements

We believe our greatest achievement of our project was the formula we used to find the y – position of the point on our graph. While it was originally conceived as a one-time use formula only to be used in this project, we discovered that it is easily adapted to other uses. We have placed it (with a few modifications) in other programs that require graphs and it made writing the program a lot easier. We did not pick our program itself to be our greatest achievement because it has been done before , and is intended to be a free alternative to other weight and balance calculators that you can get online.

Acknowledgements

Cessna Model 172N Skyhawk Information Manual. Wichita: Cessna Aircraft Company , 1978.

Lieutenant Colonel Charles Hainline (USAF)

Albert Simon , Computer Science Instructor , Alamogordo High School

Java How to Program Fourth Edition. Deitel and Deitel . Prentice Hall, Upper Saddle River , NJ. 2002.