Coordinate Conversion Library / Tool
Author: Matt Bichay
Date: Friday, October 23, 2015
https://github.com/mbichay/CoordinateTransforms

Background:

This library is based on the WGS84 Standard ellipsoid of the Earth. All inputs will assume that the original inputs are within the WGS84 standard coordinate system. Coordinate conversions are simply mathematical conveniences for doing external calculations when dealing with specific positions on Earth. For example, GPS devices use a large amount of coordinate conversions to keep track of whatever its looking at. These are also useful for looking at things in the sky, such as airplanes or satellites. For this library, we are neglecting that some of the coordinate frames are rotating frames and can be extremely powerful when looking at moving coordinates with a velocity.

Phase 1 – Back end:

First off, there must be a back end for handling a dynamic range of coordinate inputs. This requires the use of different distance metrics and the feasibility for moving between Degrees or DMS and Radians.

DistanceMetric.java: Handles metric unit conversions between Gigameters, Megameters, Kilometers, Meters, Decimeters, Centimeters, Millimeters, Micrometers, Nanometers, Picometers, and Femtometers. The value is stored in a double which can be accessed by the get/setValue methods. The unit is an enumeration which is stored in a enum Unit variable and can be accesed with the get/setUnit methods. Note: This class supports toString and deep copy clone functionalities.

- Default 0 meter constructor and basic value/unit constructor.
- Private double value;
- Private Unit unit;
- Enum Unit containing all the supported distance metrics with corresponding toString and asExponent methods.
- as(Unit type) methods for converting between distance metric units.
- getter/setter for Unit type and Value.
- Clone method
- toString method

Radians.java: Handles to radian conversions of degrees or DMS. This is a mathematical convenience for many of the coordinate transformation formulas which do calculations on the radian value of a Latitude or Longitude.

- Default 0m radians constructor, DMS conversion constructor, and radian value based on double inpuit.
- Private double radians;
- asDMS and asDecimal methods for retrieving the radian value as a DMS object or double.
- Corresponding getter/setter for the radian value.
- Clone method
- toString method

DMS.java: Handles degree conversion to/from radians and decimal values of degrees. Degrees, Minutes, and Seconds is a common way to express a Latitude or Longitude value. DMS is primarily how Lat/Lon will be stored and will also prove to be useful when doing coordinate conversions.

- Default constructor of 0,0,0, Basic int,int,double constructor, Radians conversion cosntructor.
- Private int degrees;
- private int minutes;

- private double seconds;
- getters/setters for private data members degrees,minutes, and seconds.
- AsRadians method for to radian value conversion.
- AsDecimal for converting into a 3 part D,M,S, value to a single double value degree
- Clone method
- toString method

WGS84.java: This is a class for holding public final static constants related to the WGS84 standard.

Wikipedia: The World Geodetic System (WGS) is a standard for use in cartography, geodesy, and navigation including by GPS. It comprises a standard coordinate system for the Earth, a standard spheroidal reference surface (the datum or reference ellipsoid) for raw altitude data, and a gravitational equipotential surface (the geoid) that defines the nominal sea level.

WGS84 Defining Parameters

Parameter	Notation	Value
semi-major axis	a	6378137.0 m
Reciprocal of flattening	1/*f*	298.257223563

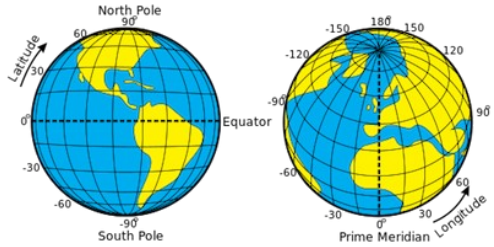
WGS84 derived geometric constants

Constant	Notation	Value
Semi-minor axis	*b*	6356752.3142 m
First Eccentricity Squared	e^2	$6.69437999014 \times 10^{-3}$
Second Eccentricity Squared	e'^2	$6.73949674228 \times 10^{-3}$

- public static final DistanceMetric EARTH_SEMIMAJOR_AXIS
- public static final DistanceMetric EARTH_RECIPROCAL_FLATTENING
- public static final DistanceMetric EARTH_SEMIMINOR_AXIS
- public static final DistanceMetric FIRST_ECCENTRICITY_SQUARED
- public static final DistanceMetric SECOND_ECCENTRICITY_SQUARED

CoordinatePosition.java: Tag interface for tagging the LLA, ECEF, and ENU classes as CoordinatePosition type objects. This is useful only if the user wants to expand upon the library and hold coordinates in an array. Ensures that there will be an asLLA, asECEF, and asENU method in all coordinate positions.

LLA.java: This is a class for holding Latitude, Longitude, and Altitude WGS84 Coordinates. Note, there is no input sanitation and assumes the user is well versed enough to understand how to input WGS84 coordinates.



- Default constructor 0,0,0 meters, Basic constructor DMS,DMS,DistanceMetric or Radians,Radians,Distance metric (Note: radians will convert to DMS before being stored).
- ECEF Conversion Constructor (Refer to the formula below)
- Private DMS lat;
- Private DMS lon;
- Private DistanceMetric alt;
- Getters/Setters for the private member variables.
- AsECEF – Calls the ECEF conversion LLA ↔ ECEF conversion constructor in ECEF.java, then returns a the object.
- AsLLA – returns itself cloned.
- AsENU(LLA refPt) - Requires a reference LLA to be passed in for conversion, passes the asECEF object into the

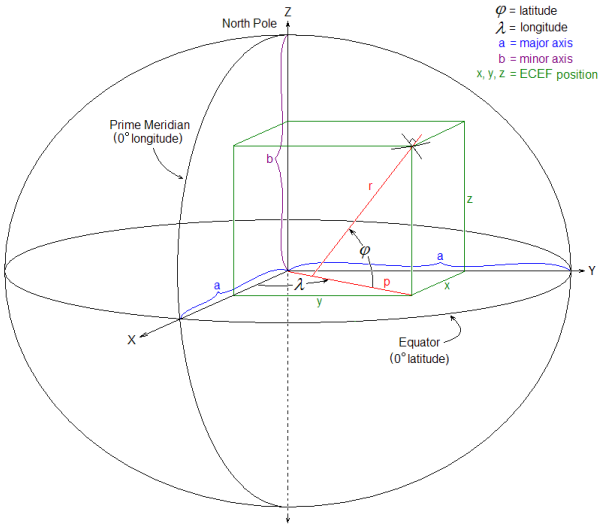
ECEF ↔ ENU constructor in ENU.java. Conversions must to ENU must go through ECEF and if there is no reference point passed as an argument, an arbitrary 45,45,Earth Surface argument will be given.

ECEF to LLA Formula and Matlab Sourcecode (Matlab referenced do to the mathematical complexity of the ECEF to LLA formula)

$$
\begin{aligned}
r &= \sqrt{X^2+Y^2} \\
E^2 &= a^2-b^2 \\
F &= 54b^2Z^2 \\
G &= r^2+(1-e^2)Z^2-e^2E^2 \\
C &= \frac{e^4Fr^2}{G^3} \\
S &= \sqrt[3]{1+C+\sqrt{C^2+2C}} \\
P &= \frac{F}{3\left(S+\frac{1}{S}+1\right)^2G^2} \\
Q &= \sqrt{1+2e^4P} \\
r_0 &= \frac{-(Pe^2r)}{1+Q}+\sqrt{\frac{1}{2}a^2\left(1+1/Q\right)-\frac{P(1-e^2)Z^2}{Q(1+Q)}-\frac{1}{2}Pr^2} \\
U &= \sqrt{(r-e^2r_0)^2+Z^2} \\
V &= \sqrt{(r-e^2r_0)^2+(1-e^2)Z^2} \\
Z_0 &= \frac{b^2Z}{aV} \\
h &= U\left(1-\frac{b^2}{aV}\right) \\
\phi &= \arctan\left[\frac{Z+e'^2Z_0}{r}\right] \\
\lambda &= \arctan 2[Y,X]
\end{aligned}
$$

```
% calculations:
b   = sqrt(a^2*(1-e^2));
ep  = sqrt((a^2-b^2)/b^2);
p   = sqrt(x.^2+y.^2);
th  = atan2(a*z,b*p);
lon = atan2(y,x);
lat = atan2((z+ep^2.*b.*sin(th).^3),(p-e^2.*a.*cos(th).^3));
N   = a./sqrt(1-e^2.*sin(lat).^2);
alt = p./cos(lat)-N;

% return lon in range [0,2*pi)
lon = mod(lon,2*pi);

% correct for numerical instability in altitude near exact poles:
% (after this correction, error is about 2 millimeters, which is about
% the same as the numerical precision of the overall function)

k=abs(x)<1 & abs(y)<1;
alt(k) = abs(z(k))-b;

return
```

ECEF.java: Earth-Centered, Earth-Fixed is a coordinate frame of which the reference point of the coordinate is in reference to the center of mass of our Earth. It is noted as an X,Y,and Z position where Z is the north pole and X and Y point towards different positions on the equator.



- Default constructor of 0,0,0. Basic constructor of Distance,Distance,Distance
- From LLA conversion constructor. Note: see formula below.
- From ENU conversion constructor. Note: see formula below.
- DistanceMetric xPos, yPos, and zPos;
- Appropriate getters/setters for the private member variables
- asENU method (requires a reference point) calls the From ECEF ENU.java constructor. If no reference point is given, it will use an arbitrary 45,45,Earth surface reference point.
- AsLLA method calls the from ECEF conversion constructor in LLA.java and returns an LLA object.
- AsECEF will return as copy of itself.
- toString method
- Clone method

LLA to ECEF Formula

Coordinate Conversion
Geodetic Latitude, Longitude, and Height to ECEF, X, Y, Z

$$X = (N + h)\cos\phi\cos\lambda$$
$$Y = (N + h)\cos\phi\sin\lambda$$
$$Z = [N(1 - e^2) + h]\sin\phi$$

where:

ϕ, λ, h = geodetic latitude, longitude, and height above ellipsoid

X, Y, Z = Earth Centered Earth Fixed Cartesian Coordinates

and:

$N(\phi) = a / \sqrt{1 - e^2 \sin^2\phi}$ = radius of curvature in prime vertical

a = semi-major earth axis (ellipsoid equatorial radius)

b = semi-minor earth axis (ellipsoid polar radius)

$$f = \frac{a - b}{a} = \text{flattening}$$

$e^2 = 2f - f^2$ = eccentricity squared

Peter H. Dana 8/3/96

ECEF to ENU Formula

Transforming ENU to ECEF

Let us consider a point *Q* that has the coordinates *(x, y, z)* in the ECEF system and coordinates *Q' = (e, n, u)* in the ENU system located at *P = (P_x, P_y, P_z)*. The spherical coordinates of *P* are *(φ, λ, R)*. Then the following equation transforms *(e, n, u)* to *(x, y, z)*:

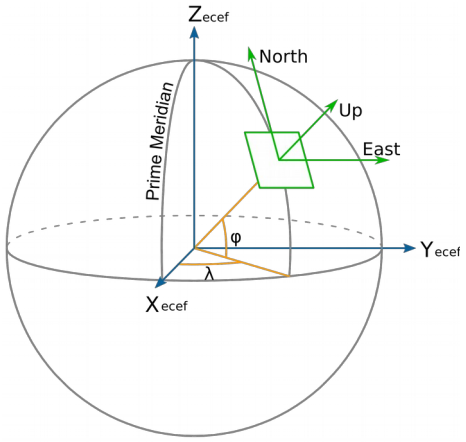
$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} -\sin\lambda & -\sin\varphi\cos\lambda & \cos\varphi\cos\lambda \\ \cos\lambda & -\sin\varphi\sin\lambda & \cos\varphi\sin\lambda \\ 0 & \cos\varphi & \sin\varphi \end{bmatrix} \begin{bmatrix} e \\ n \\ u \end{bmatrix} + \begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix} \quad (10)$$

Notice that the columns of the transformation matrix ***M*** are equal to the vectors ***E***, ***N***, ***U***, respectively. Hence, the matrix ***M*** is orthonormal. It corresponds to the rotation of the XYZ-System around the origin into the ENU-System (centered at origin). The transformation is built by first rotating the vectors ***X***, ***Y***, ***Z*** around the origin into ***E***, ***N***, ***U***, and then translating this system by the vector ***P*** into its final position.

The transformation can be written concisely using vector and matrix notation:

$$Q = M\,Q' + P \quad (11)$$

ENU.java: East, North, Up is a coordinate frame from the perspective of any dynamic reference point. It looks at coordinates from an east,north, then directly up perspective of the reference. Choosing a good reference is the key to the usefulness of this coordinate transform, typically its where you're looking from. Useful for looking at objects above you, such as an airplane or satellite.



- Default constructor 0,0,0, Basic constructor Distance,Distance,Distance.
- From ECEF to ENU conversion constructor. See formula below.
- DistanceMetric ePos;
- DistanceMetric nPos;
- DistanceMetric uPos;
- Corresponding getters/setters for the East, North, and Up data members.

- AsENU Method, returns clone of itself.
- AsECEF method, Requires a reference point, if none given, uses a 45,45,Earth surface default reference point. Calls the ENU to ECEF constructor in ECEF.java and returns an object of ECEF.
- AsLLA method, Requires a refertence point, if none given, uses a 45,45, Earth surface default reference point. Calls the asECEF method and passes the ECEF object into the ECEF to LLA constructor in LLA.java. Returns an object of LLA.
- Clone method
- toString method

ECEF to ENU Constructor (Note requires a reference point).

3. By means of a rotation, displacements in ECEF coordinates are transformed to ENU coordinates.
The ECEF coordinates (dx, dy, dz) are orientated in such a way that from the centre of the Earth the $\hat{z}$ points in the direction of true north, $\hat{x}$ points in the direction of the prime meridian and the direction of $\hat{y}$ is 90° from the prime meridian, see figure 1. The orientation of ENU coordinates is determined by rotating the ECEF coordinates; firstly about the $\hat{z}$ axis by λ degrees and then the new $\hat{y}$ axis by ϕ degrees,

$$\begin{pmatrix} de \\ dn \\ du \end{pmatrix} = \begin{pmatrix} -\sin\lambda & \cos\lambda & 0 \\ -\sin\phi\cos\lambda & -\sin\phi\sin\lambda & \cos\phi \\ \cos\phi\cos\lambda & \cos\phi\sin\lambda & \sin\phi \end{pmatrix} \begin{pmatrix} dx \\ dy \\ dz \end{pmatrix}. \quad (3)$$

Substituting equation (2) into equation (3), and ignoring terms of $\mathcal{O}(d\theta^3)$ and $\mathcal{O}(dhd\theta^2)$ and higher, we get

$$\begin{aligned} de &= \left(\frac{a}{\chi} + h\right)\cos\phi\, d\lambda - \left(\frac{a(1-e^2)}{\chi^3} + h\right)\sin\phi\, d\phi\, d\lambda + \cos\phi\, d\lambda\, dh \\ dn &= \left(\frac{a(1-e^2)}{\chi^3} + h\right) d\phi + \tfrac{3}{2}a\cos\phi\sin\phi e^2 d\phi^2 + dh\, d\phi \\ &+ \tfrac{1}{2}\sin\phi\cos\phi\left(\frac{a}{\chi} + h\right) d\lambda^2 \quad (4) \\ du &= dh - \tfrac{1}{2}a\left(1 - \tfrac{3}{2}e^2\cos\phi + \tfrac{1}{2}e^2 + \tfrac{h}{a}\right) d\phi^2 \\ &- \tfrac{1}{2}\left(\frac{a\cos^2\phi}{\chi} - h\cos^2\phi\right) d\lambda^2 \quad . \end{aligned}$$

Note:

The coordinates (ϕ, λ, h) are ellipsoidal (WGS84), not spheroidal (geocentric). The difference is most easily explained by figure 2.

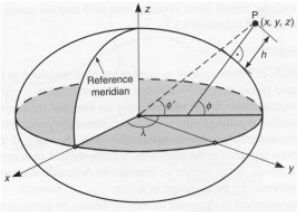


Figure 2: Ellipsoidal and spheroidal coordinates

Phase 2 – Front end:

CoordinateTransformGUI.java: A Swing/AWT GUI for the small little conversion tool.

- From comboBox and toCombo box with the LLA,ECEF,ENU selectables.
- Input panel for inputting the LLA,ECEF,ENU from coordinates and the reference point if neccicary.
- Convert button
- Results text box underneath the convert button.
- SetInputPanel method for converting the input panel to accommodate the different combinations of FROM and TO conversion options.
- SetConverText method for displaying the results of the conversions as a string.
- Add Combobox and Button listener methods for the controller to connect.
- GetStandardInput and getRefPtInput methods for retrieving the data in the input fields as doubles.

CoordinateManager.java: A wrapper and delegator for controlling the from ↔ to conversions in the CoordingateTransform tool. This is the secondary hook in for the controller.

- Default constructor for initializing the private data members.
- Private LLA lla;
- Private ECEF ecef;
- Private ENU enu;
- Private LLA refPt;
- lla, enu, RefPt, lla2ecef, lla2enu, ecef2lla, ecef2enu, enu2lla, enu2ecef methods for performing the conversions and returning the toString methods of the corresponding conversions.

CoordinateTransformController.java: Controller for the model (CoordinateManager) and the View (CoordinateTransformGUI). Contains the comboBox change listener and logic for the comboBoxes and also contains the convertButton listener and controls the logic for the convert buttons. This attaches itself to the model via listeners and controls the models via the listener actions.

- Default constructor, connects the view and the manager to itself. It also sets up the action listeners for the view components.
- Public CoordinateTransformGUI view;
- Public CoordinateManager model;
- convertButtonListener – retrireves the values from reference/standard inputs of the view, sanitizes the view for incorrect inputs, and then goes through a conditional tree to select the correct CoordinateManager function to display in the view conversion result text box.
- ComboBoxListener – controls the view and swaps the input panels according to what the user's selection for the From and To combo boxes.

Driver.java: Simple driver for the program, creates a view, creates a model, and links them to the newly created controller.

Testing: Testing can be done by inputting a correct WGS84 Standard Latitude, Longitude, and Altitude value in meters. This can be done with the link provided below, simply choose a place on the Google map and paste the corresponding LLA. Results can be checked also using the link below for ECEF. To test ENU takes a little bit of intuition, either you must have a copy of Matlab and do your own testing based upon the built-in matlab functions, or choose a reference point and use your intuition to choose a LLA coordinate in a good position/viewpoint from your reference point. Then estimate your E,N,U distances in meters and see if you can come up with a reasonable answer. The final portion of the testing is to actually go backwards from each point and see if you can get similar answers.

Note: Results can vary based upon reference points, please choose one which makes sense (IE: Not something directly on the Z axis of the globe)

It is common to loose some precision during the conversions, usually they are out 5 – 10 decimal places and are a result of using basic, non-industry spec conversion methods.

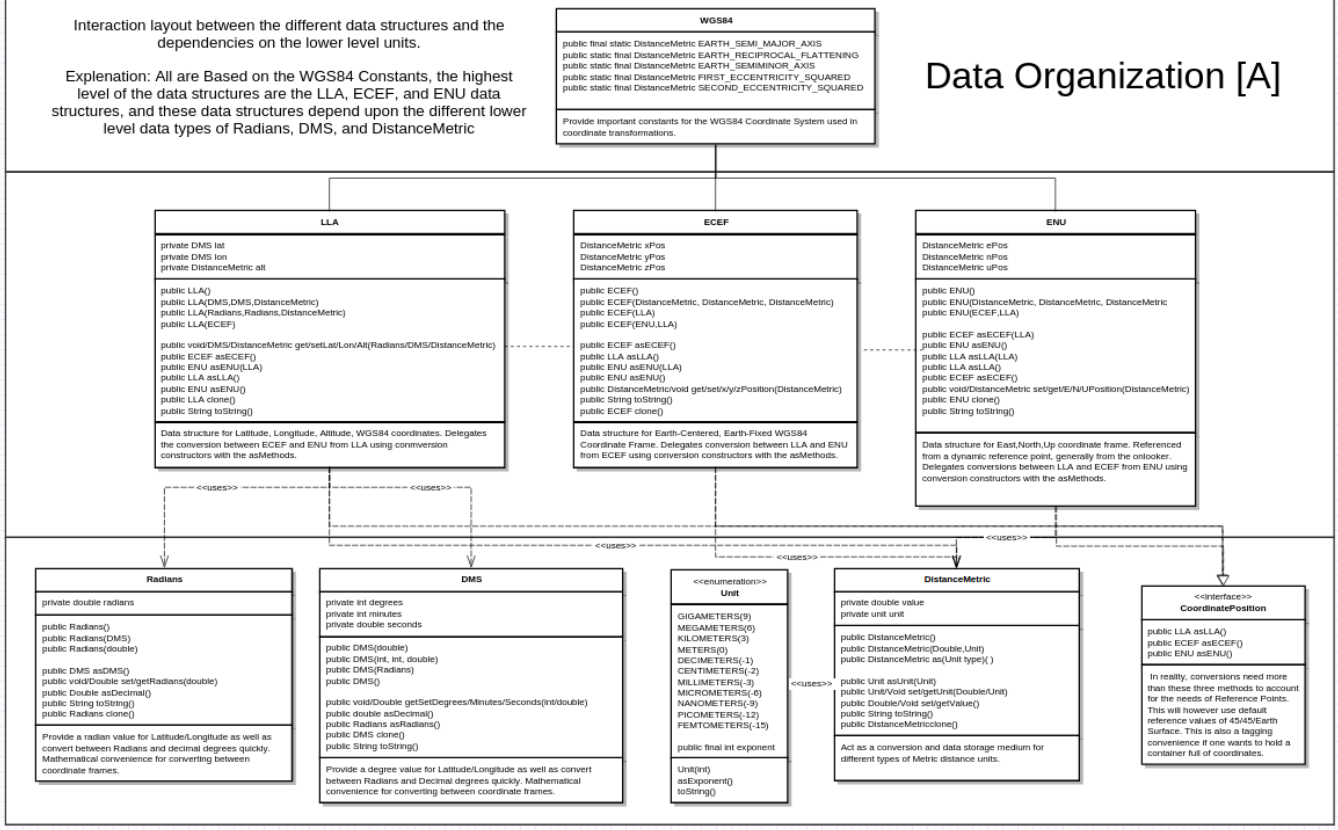
http://www.mapcoordinates.net/en
http://www.oc.nps.edu/oc2902w/coord/llhxyz.htm

UML:

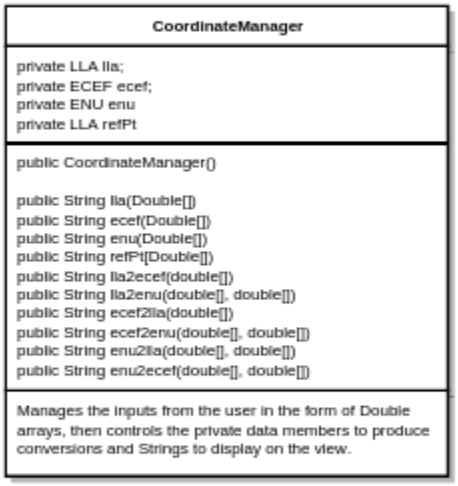
Interaction layout between the different data structures and the dependencies on the lower level units.

Explenation: All are Based on the WGS84 Constants, the highest level of the data structures are the LLA, ECEF, and ENU data structures, and these data structures depend upon the different lower level data types of Radians, DMS, and DistanceMetric

Data Organization [A]

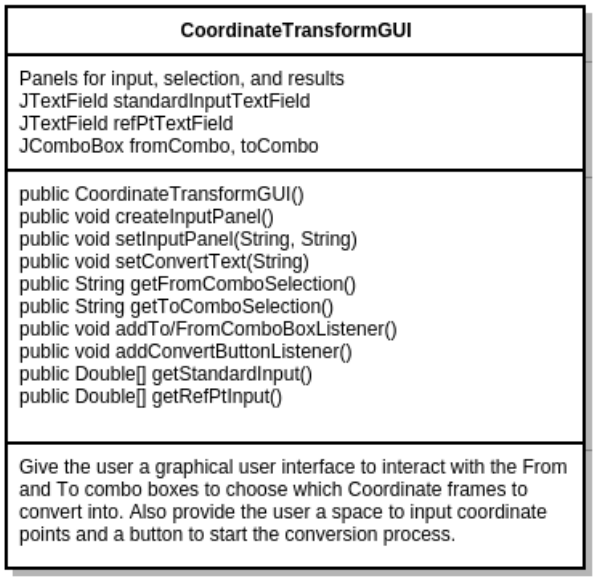


The Model [B]

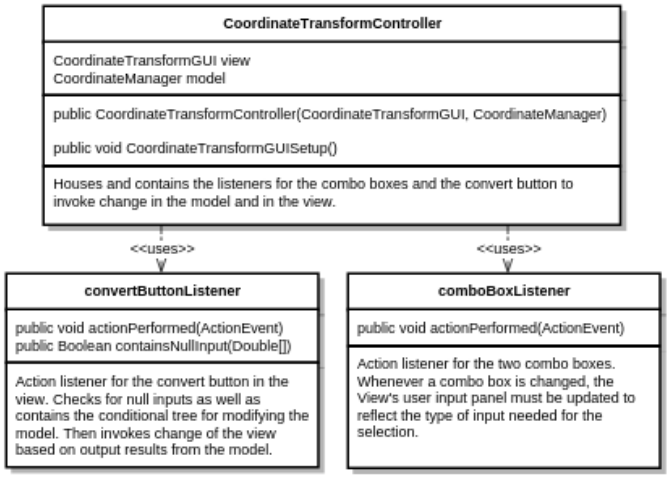


[A]

The View [C]



The Controller [D]



[B] & [C]

Screenshots: