**Celestial Distance Calculator – LeAnne M. Branch**

**Github link:** <https://github.com/branchwag/celestialDistanceCalc>

**Project Writeup**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Introduction:**

The Celestial Distance Calculator is an interactive Java application designed to visualize a section of the night sky and demonstrate how astronomers calculate distances to various celestial objects. This educational tool allows users to explore multiple distance calculation methodologies that astronomers use, from the relatively straightforward parallax method for nearby stars to the redshift-based calculations for distant galaxies.

The application bridges theoretical astronomical concepts with a practical, hands-on interface to make these complex calculations more accessible and understandable. Users can select celestial objects on a simulated star map and view detailed information about the object along with step-by-step distance calculations.

Please note that Stars have two calculation methods in this program – the Parallax Method and the Spectroscopic Parallax Method. Galaxies have one method of calculation - Redshift.

**Sources:**

David Butler ‘How Far Away Is it?’ Series:  
<https://www.youtube.com/watch?v=HgNJwg2GISs&list=PLpH1IDQEoE8QWWTnWG5cK4ePCqg9W2608>

Chapter 5, Nearby Stars, provides and explains the Parallax calculations used for ‘nearby’ stars (starting at 1:45).

Chapter 6, Distant Stars, covers Spectroscopic Parallax.

Chapter 16, The Cosmos, covers distance calculations for Galaxies.

Data used in celestial object calculations is obtained from the SIMBAD Astronomical Database:   
<https://simbad.u-strasbg.fr/simbad/>

**Project Structure**

The project follows an object-oriented architecture with a separation between the model and view components.

**Class Hierarchy**

The application is structured with the following key classes:

1. **CelestialDistanceCalculator**: Main application class that initializes the program.
2. **MainFrame**: The main application window that organizes the GUI components.
3. **CelestialObject** (abstract): Base class for all celestial objects with common properties.
   * **Star**: Extends CelestialObject with star-specific properties and distance calculation methods.
   * **Galaxy**: Extends CelestialObject with galaxy-specific properties and distance calculation methods.
4. **SkyMap**: Manages the collection of celestial objects and handles object selection.
5. **StarFieldPanel**: Displays the interactive star field with celestial objects.
6. **ObjectInfoPanel**: Displays detailed information and calculations for the selected object.

**Key Features and Implementation**

**Interactive Star Field**

The interactive star field is implemented in the StarFieldPanel class. This panel:

* Renders a coordinate grid system based on Right Ascension (hours) and Declination (degrees)
* Displays celestial objects at their approximate positions
* Captures mouse events to detect object selection
* Creates a background of smaller, random stars to enhance visual appeal

**Object-Oriented Celestial Object Representation**

The application uses polymorphism to model different types of celestial objects with specific behaviors while maintaining a common interface.

The abstract CelestialObject class defines common properties and behaviors:

* Name, coordinates, apparent magnitude, and color
* Abstract methods for distance calculation and information display
* Concrete methods for drawing and coordinate transformation

The Star and Galaxy classes extend this base class with specialized implementations:

**Star Class Features:**

* Includes properties like parallax, absolute magnitude, and spectral type
* Implements two distance calculation methods:
  + Parallax method: distance = 1 / parallax (converted to light years)
  + Spectroscopic parallax: Using the relationship between apparent and absolute magnitude

**Galaxy Class Features:**

* Includes properties like redshift and galaxy type
* Implements distance calculation using Hubble's Law:
  + Velocity calculation from redshift: v = c \* z
  + Distance calculation: d = v / H₀
  + Conversion from megaparsecs to millions of light years

**Distance Calculation Methods**

**For Stars:**

**Parallax Method** (for nearby stars):

distance (in parsecs) = 1 / parallax (in arcseconds)

distance (in light years) = distance (in parsecs) \* 3.26

**Spectroscopic Parallax Method** (for stars beyond parallax range):

5 \* log10(d) = m - M + 5

where:

d = distance in parsecs

m = apparent magnitude

M = absolute magnitude

For example, Alpha Centauri A has the following properties as per SIMBAD:

<https://simbad.u-strasbg.fr/simbad/sim-basic?Ident=Alpha+Centauri&submit=SIMBAD+search>

A computer screen shot of a program

AI-generated content may be incorrect.

This means:  
  
Right Ascension: 14h 39m 36.50s,

Declination: -60°50'02.30",

Visual magnitude: -0.01,

Parallaxes (mas): 750.81 [0.38] B 2021AJ....162...14A

Absolute magnitude (SIMBAD gives apparent mag and parallax which can be used to calculate this): 4.38,

Spectral type: G2 V

In the SkyMap class, within the initCelestialObjects function, this is represented as below, accounting for rounding:

celestialObjects.add(new Star("Alpha Centauri A", 14.66, -60.83, 0.01, 0.75, 4.38, "G2V"));

Constructor from Star.java for reference:

public Star(String name, double rightAscension, double declination, double apparentMagnitude, double parallax, double absoluteMagnitude, String spectralType) {…

\*Note that Right Ascension value was derived by converting the given RA in decimal hours to hours, like so:  
  
14 hours provided.

Then, convert given minutes (39) into fractional hours: (39/60 = 0.65 hours).

Then, convert the given seconds into fractional hours: 36.50 / 3600 = ~0.010138 hours.

Finally, add all components. 14 + 0.65 + 0.010138 = ~14.660138 hours.

Round the result to 14.66.

\*As for the absolute magnitude of 4.8:

Parallax is provided as 750.81

Apparent magnitude is provided as V = -0.1

M = m + 5 – 5 log10(d)

Where:

M = absolute magnitude

m = apparent magnitude

d = distance in parsecs

distance = 1000 / 750.81 = 1.332 parsecs

M = -0.1 + 5 – 5log10(1.332)

= 4.2765

Round up to 4.28.

As a ‘day 2’ enhancement to this calculator, it could be adjusted to take RA value as provided rather than having the manual calculation conversion required to input the Object in code. I was also thinking that I could maybe pull in an apparent magnitude from an API for verification (NASA, for example, has a great range of APIs available).

Please note though that due to the nature of these calculations, there is still room for debate on the results!

Betelguese, for example, is a star where the distance calculation varies from 500 to 700 light years away. Using both the parallax and spectroscopic parallax methods, this program does produce results in that range, but just be aware that these results are not exact.

**For Galaxies, the redshift method was used:**

**Redshift Method** (using Hubble's Law):

velocity = speed of light \* redshiftdistance (Mpc) = velocity / Hubble constantdistance (millions of light years) = distance (Mpc) \* 3.26

**Visual Representation of Different Object Types**

The application visually distinguishes different types of celestial objects:

* **Stars** are colored based on their spectral type, matching the standard astronomical classification:
  + O-type stars: Blue
  + B-type stars: Blue-white
  + A-type stars: White
  + F-type stars: Yellow-white
  + G-type stars: Yellow
  + K-type stars: Orange
  + M-type stars: Red
* **Galaxies** are drawn with shapes corresponding to their morphological type:
  + Spiral galaxies: Circle with an outer ring
  + Elliptical galaxies: Oval shape
  + Irregular galaxies: Simple circle

The size of each object is determined by its apparent magnitude, with brighter objects appearing larger on the display.

**Program Flow**

**Program Initialization**

1. The main class CelestialDistanceCalculator creates the MainFrame
2. MainFrame initializes the model (SkyMap) and view components (StarFieldPanel and ObjectInfoPanel)
3. SkyMap populates itself with a predefined list of celestial objects
4. StarFieldPanel sets up mouse listeners and draws the initial view

**User Interaction Flow**

1. **User clicks on a celestial object** in the StarFieldPanel
2. The panel's mouse listener determines the nearest celestial object
3. The selected object is updated in the SkyMap
4. The ObjectInfoPanel is notified and updates its display
5. The panel displays:
   * Object information (name, coordinates, magnitude, etc.)
   * Available calculation methods based on object type
6. **User selects a calculation method** from the dropdown
7. The calculation is performed and displayed with step-by-step details

**Technical Challenges and Solutions**

**Converting Celestial Coordinates to Screen Coordinates:**

**Challenge:** Transforming the equatorial coordinate system (Right Ascension and Declination) to screen coordinates.

**Solution:** A linear mapping function was implemented in the CelestialObject class that converts:

* Right Ascension (0-24 hours) to X-coordinates (0 to panel width)
* Declination (-90 to +90 degrees) to Y-coordinates (panel height to 0)

**Object Selection in a Dense Field:**

**Challenge:** Determining which object the user intended to select when clicking on the screen.

**Solution:** Implemented a nearest-object algorithm in the SkyMap class that:

* Calculates Euclidean distance between click location and each object
* Returns the closest object within a specified threshold
* Sets a reasonable threshold to balance between precision and ease of selection

**Visual Representation of Different Object Types:**

**Challenge:** Visually distinguishing different types of celestial objects.

**Solution:** Custom drawing methods in each subclass:

* The Star class uses color mapping based on spectral type
* The Galaxy class draws different shapes based on galaxy type
* Object size is scaled based on apparent magnitude

**Future Enhancements**

1. **Calculation Methods:**
   * Eliminating manual calcs from sources/pulling directly from source via API
   * Cepheid variable method for intermediate distances
   * Type Ia supernovae method for very distant galaxies
   * Expanding the cosmic distance ladder representation
2. **User Interface Improvements:**
   * Zoom capability to explore dense regions
   * Ability to ‘pan around’ to new regions of sky
   * Search functionality to find specific objects
   * Time controls to show object movement over time
   * Light curve visualization for variable stars
3. **Data Expansion:**
   * Ability to import catalogs of celestial objects
   * Displaying additional properties like radial velocity
   * Adding more object types (quasars, nebulae, etc.)
4. **Educational Features:**
   * Interactive tutorials explaining each calculation method (maybe tooltips)
   * Historical context about the development of distance measuring techniques
   * 3D visualization

**Conclusion**

The Celestial Distance Calculator was a fun tool to build! Though I am not a professional astronomer by any means, I have always been very much interested in space (as you can likely see from some of the other projects on my Github). Getting the SkyMap/StarField working was my favorite part, because then I could visualize the end product much better. Once I resolved a few nasty bugs (including one involving the background stars), I feel like it ended up looking rather good!

The interface is relatively simple in my opinion (perhaps for the better), but the complexity reveals itself in the code (especially for the calculations). Perhaps one day I can do a 3D rendition of this (perhaps using NextJS/ThreeJS as I am still a bit more comfortable in Javascript than Java at the moment). For now though, I am very happy that the stars are only a mouse click away.