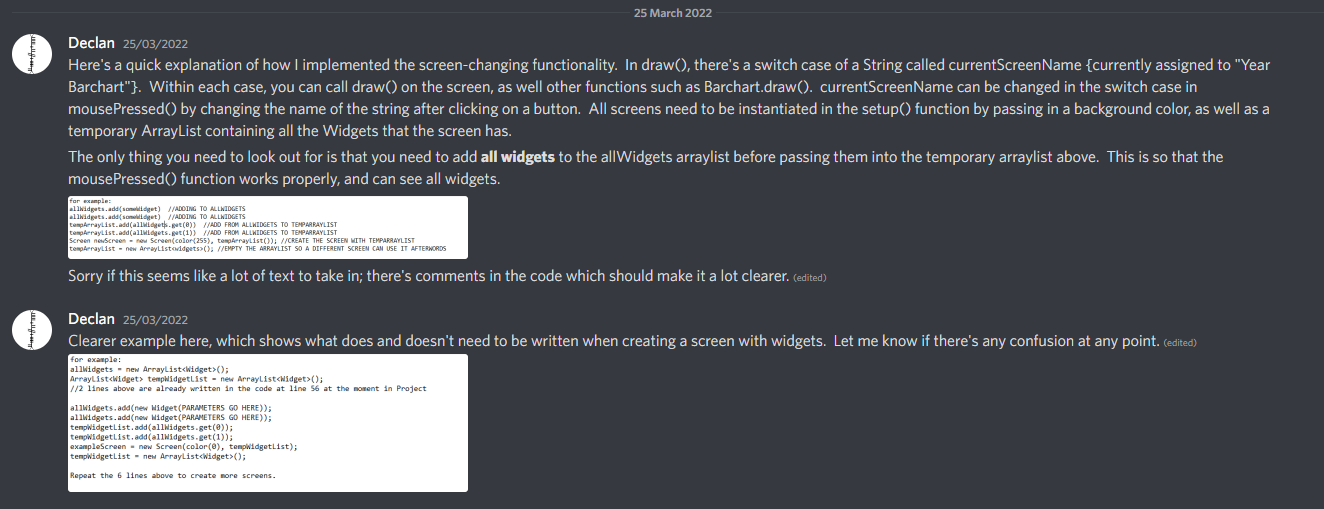
# Programming Project - Group 8

Declan Michael McCabe + Mia Cimas + Darius Pop + Kostiantyn Ohorodnyk

## Introduction:

Our team consists of Declan McCabe, Mia Cimas, Darius Pop and Kostiantyn Ohorodnyk. We communicated using a Discord server and also had in-person meetings every Tuesday after the lectures and online meetings on Wednesdays. In those meetings we would decide on weekly goals, and split up the work amongst ourselves, in order to make sure we had most of the work done before the start of the Thursday labs. Over the weekends, we worked on finishing off and polishing the work we had done in that week. We documented our code using an “explaining-code” channel in Discord, where we would tell the rest of the team how to implement and work with the code we have written. Below is one such example of documentation from that channel.



## Screen Organisation:

The screen class written by Declan determined a lot of how we would structure our program. This class is used to organise each screen’s widgets and the background colour. The class also checks if the mouse is hovering over any of the widgets, changing the cursor image from an arrow to a hand if that is the case.

The setup for each Screen takes place in the setup() function, called at the start of the file, where an ArrayList of the widgets is passed into the Screen’s constructor. A switch statement in the draw() function determines which screen should currently be drawn. It is at this stage where we first noticed a problem in our implementation. As convenient as it is to have the Screen handle the drawing of the widgets, it does have its limitations in terms of control over what order items are drawn in. For this reason, Declan decided to add support for widgets being added to a screen, but drawn somewhere that isn’t the Screen’s own draw() method. This allowed us to have much more control of the layers on which widgets were drawn, which was particularly helpful in the Statistics Screen. A simple event handler in the mousePressed() function was used to take care of screen switching, as well as handling user queries.

To create return buttons, Mia created a PhotoWidget subclass of Widget. The subclass inherits all instance variables and methods from its superclass and overrides the draw() method by displaying a selected picture instead of a default rectangular button.  


## Start Screen

When we first started the project, we were more focused on the functionality of the program than on its visual aspects. Around week 4, after we finished implementing all the queries, Darius and Mia decided to focus on creating a more visually appeasing layout and colour palette. They worked on the design of the start screen and added pictures, widgets, and lines to make it look more topic related. All the created widgets open different screens, which present data in various ways.

## Sort by Apogee/Perigee Screens:

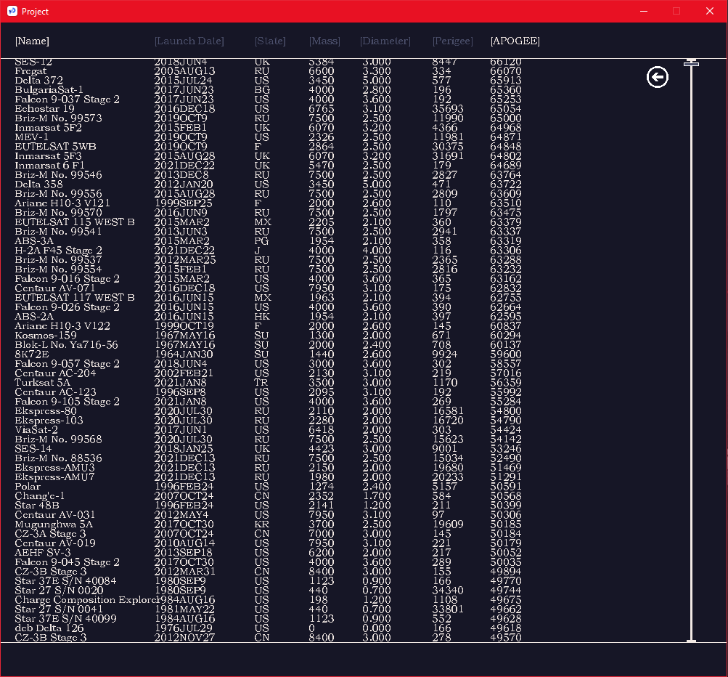
The first query we implemented was a simple list of the data, sorted by perigee or apogee. This screen was created by Declan. The data was loaded from the file using the loadStrings() method in processing. This array was then passed into a method called generateObjects(), which created an ArrayList of “spaceObjects” containing the data for each line.

The ArrayList of objects is passed into a method called sortSpaceObjects(). This method implements the method Collections.sort(), along with custom Comparator subclasses to sort the objects by various properties, which can be found int java.util.Collections and java.util.Comparator respectively.

The UI itself is drawn by a method called drawSortedData(), which displays headers at the top of the screen, and draws the properties of each spaceObject. This screen makes use of the Slider class created by Mia. The slider consists of the track, two lines to indicate its ending and beginning, and the thumb used to indicate our position on the track.



The thumb moves when the mouse is pressed and dragged along the track or if the mouse is pressed somewhere within the slider bounds. It is used on multiple screens to aid displaying large data sets by allowing the user to scroll through a list.

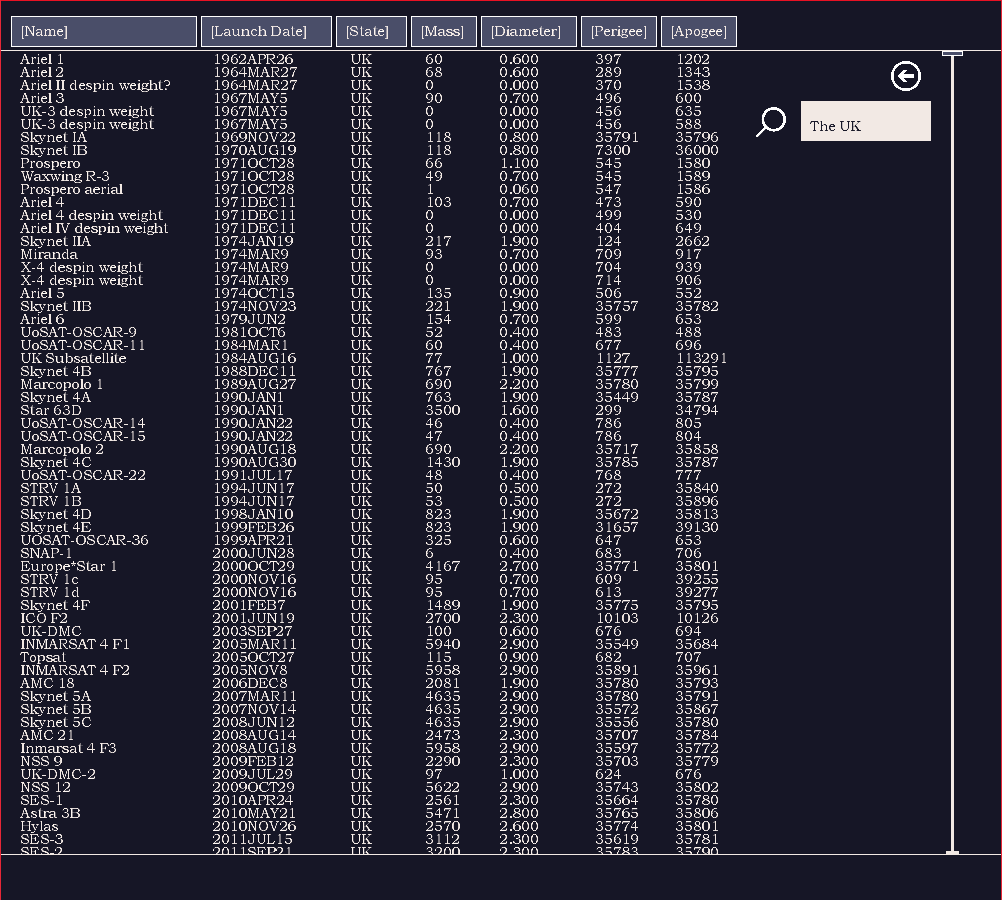


Initially, the drawSortedData() method called draw() for the entire list of objects, which was a simple solution, but led to performance issues when working with larger files. In order to fix this, it was made so that only the lines with a Y-position within the bounds of the screen had the draw() method called for them.

## Search by Country Screen:

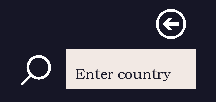
In this screen, the user is able to search for, and scroll through information pertaining to, objects of a particular country. To get user input, Darius, Kostiantyn and Declan created and implemented a search-box widget, which allowed the user to type in their query. Declan created a method called getObjectsByCountry(), which returns a subset of the full object list, only including the objects in the given country. This method supports searching by the country abbreviation (eg: UK, US, RU, CN) and the country’s actual name (eg: Russia, France, Germany). Common variations on country names are also supported (eg: The United States, The USA, The US).

To increase user-interaction, Declan implemented sorting in this screen too. The user can click on the various headers to sort the selection of objects by various parameters. This required an extension in the functionality of the sortSpaceObjects() method, such as sorting alphabetically and sorting by date. Clicking on the headers multiple times toggles between sorting high-to-low and low-to-high.



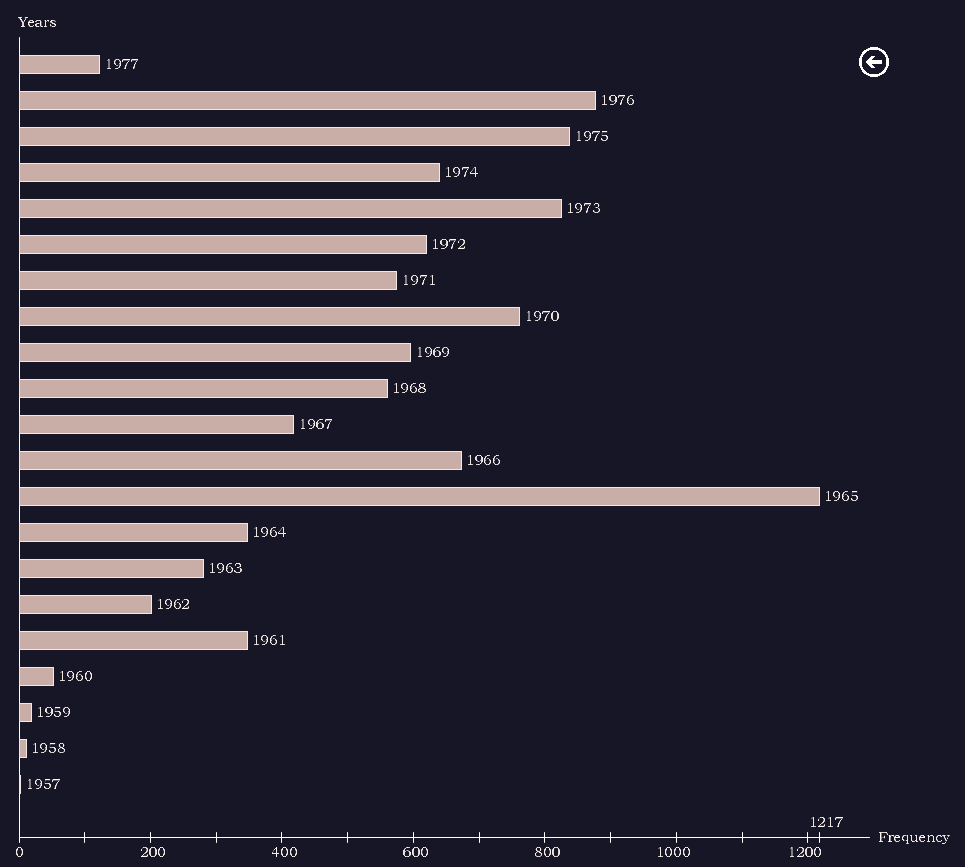
This screen makes use of the same drawSortedData() method as in the Sort by Apogee/ Perigee screens, allowing us to save on code duplication.

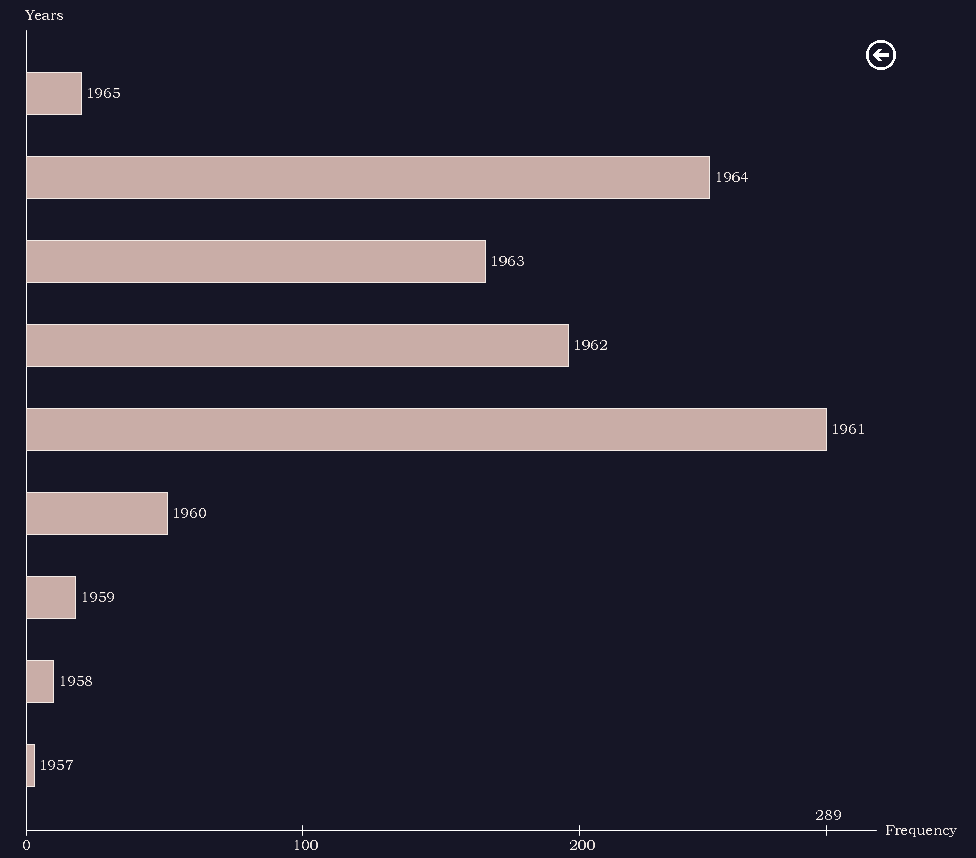
Other features were also introduced to make the UI seem more professional. Kostiantyn made it so that if no objects were being drawn (such as when the screen is first loaded, or if a search query returns empty), the slider wouldn’t appear on the screen. This was done by checking the length of the searchedObjects ArrayList and hiding the slider if it was equal to zero. Darius also made it so that if a search query was made while the slider was already scrolled down, it would jump back to the top of the list; a feature which makes the design much more intuitive. **[[DARIUS ADD SLIDER RESET STUFF HERE]]**

Finally, Mia used the PhotoWidget subclass to create a magnifying glass search widget which can be used to submit user input instead of pressing the enter key.  


## Year Bar Chart Screen:

The YearBarChart class is used to create bar charts, which show the number of launched space objects for each year within an interval (e.g., 1967 – 1987). To evenly space out the bars, Mia used a scale, which made it possible to draw any number of bars spaced out evenly on the y axis. As for the frequency points, if the highest number of launches is a four-digit number, the code displays the frequency points as multiples of 200 (e.g., 200, 400, 600, 800 etc.), otherwise it shows the frequency points as multiples of 100 (e.g., 100, 200, 300, 400 etc). Finally, the animation is achieved by incrementing a temporary width variable of the bars every time the draw method is called. The temporary variable stops incrementing when it reaches its highest width value for a certain bar.





## Statistics Screen:

The statistics screen is a screen implemented by Declan which gives some brief summary statistics about the collection of space objects. To make this screen more visually interesting, the UI was made to look like the inside of a spaceship, with shooting stars visible outside the windows, and a statistics display made to look like a semi-transparent, futuristic computer terminal projected onto the window with green text.

To achieve this look, a PNG image of the inside of a spaceship was found online, and paint.net was used to erase the outside of the ship, leaving only the stars and planets. A ShootingStar class was also created by Declan, which moves a small white circle back and forth randomly across the screen. Since the PNG was made transparent in some areas, the stars pass behind the image when moving across the planets and parts of the ship, giving the appearance that they are outside of the ship. This was an instance where we had to make us of the feature mentioned in the “Screen Organisation” section for drawing widgets outside of the Screen’s draw() method, since we needed complete control over what layers objects were being drawn on. Otherwise, we would’ve had trouble making sure that the shooting stars moved behind the correct objects. The statistics display was made to look semi-transparent by editing the alpha value of the fill colour.

The statistics were calculated using a class written by Mia called SummaryStatistics, as well as a couple of extra statistics calculated by Declan. The buttons below the window allow the user to choose what statistics are shown. 

## Polishing + Efficiency + Finishing Touches:

[MAYBE PUT THESE IN THE CORRESPONDING SCREEN HEADERS]

In the last week, Mia organised the code to make it easier to follow by adding a “constants” tab, deleting unused code which was not specified by the weekly goals, and modifying certain variable names.