Object Oriented Programming in Java

Project Milestones

Table of Contents

[1. Introduction 2](#_Toc157634433)

[1.1. Learning Goals 2](#_Toc157634434)

[2. Group Members 3](#_Toc157634435)

[3. Declaration of Authorship 4](#_Toc157634436)

[4. Milestones 6](#_Toc157634437)

[4.1. Milestone 1 6](#_Toc157634438)

[4.1.1. Things that have been achieved so far 6](#_Toc157634439)

[4.1.2. Things we are planning to achieve until the next submission 7](#_Toc157634440)

[4.1.3. Problems that arose during the last milestone phase 7](#_Toc157634441)

[4.2. Milestone 2 8](#_Toc157634442)

[4.2.1. Things that have been achieved since the last submission 8](#_Toc157634443)

[4.2.2. Things we are planning to achieve until the final submission 8](#_Toc157634444)

[4.2.3. Problems that arose during the last milestone phase 8](#_Toc157634445)

[5. Figures 9](#_Toc157634446)

[6. Technical description 13](#_Toc157634447)

[7. User Handbook 14](#_Toc157634448)

# Introduction

In the history of mankind, there have been countless inventions that fundamentally changed the way we as humans have lived. One such invention, the airplane invented in 1903, allowed us, for the first time ever, to take to the skies.  
Over a century later, we can now travel thousands of miles or fly hundreds of tons of cargo from one end of the world to the other within a day.

As part of our **Java-OOP project** we are delighted to get to know a little bit more about drones and build an application which fetches data from a Django based web server and visually displays it.  
Our aim is to create a working app using the fundamentals of object-oriented programming in Java. The app should allow for user interaction with said data, using buttons and multiple windows which ensure that the app does not look cluttered.   
The Java Framework known as “Swing” is used for front end development.

Interestingly, the rendered data also contains dynamic data which changes in regular intervals. This makes the application standout in terms of data-rendering. The written code should deal with the server’s restrictions and access the server’s data using a basic authentication header with an access token.  
Additionally, we made sure that our application does not send too many requests at once, which could lead to the application or worse the server temporarily not working.

## Learning Goals

After successfully completing this project, our team will have learnt…

1. How to coordinate in a team
2. How to fairly and efficiently divide tasks among a group
3. How to solve problems together ensuring smooth development
4. How to come up with and implement a feature or in case we lack the knowledge research a solution as a group

# Group Members

|  |  |
| --- | --- |
| **Responsibilities** | **Name** |
| Frontend Team | Mohit, Bilal, Utkarsh |
| Backend Team | Andrej, Yun See |
| Designing Team | Whole Group |
| GitHub Management | Andrej und Utkarsh |

\*\*Besides each member’s individual work on the project, we also held group meetings at least once per week in which we discussed the project and shared our opinions on the code and how to improve it. These discussions were not limited to our “own section” of the code despite the group’s designation of responsibilities. \*\*

# Declaration of Authorship

Ein Bild, das Text, Brief, Schrift, Dokument enthält.

Automatisch generierte Beschreibung Ein Bild, das Text, Brief, Dokument, Schrift enthält.

Automatisch generierte Beschreibung Ein Bild, das Text, Brief, Schrift, Dokument enthält.

Automatisch generierte Beschreibung Ein Bild, das Text, Brief, Schrift, Dokument enthält.

Automatisch generierte Beschreibung Ein Bild, das Text, Brief, Handschrift, Schrift enthält.

Automatisch generierte Beschreibung

# Milestones

## Milestone 1

## Things that have been achieved so far

1. Visual representation (wireframe) of the App using Balsamiq  
   (Discussed collectively in 2 meetings)
2. UML diagram of the back end  
   (Andrej and Yun See)
3. Back end:
   1. Data can be successfully fetched from the API and formatted according to our needs  
      (Andrej and Yun See)
4. Front-end:
   1. Basic windows with graphics and images which outline the application’s general structure
   2. Basic buttons with basic functionality such as opening a new sub window
   3. Team decided to learn tools of swing first

(Mohit, Bilal, Utkarsh)

\*\*Apart from that, a version control of the application has been initialized using a private repository of GitHub and access to the repository is confined to group members only. Various branches are used to develop individual features and then merged to the master branch of the repository \*\*

## Things we are planning to achieve until the next submission

1. Front end:
   1. Successfully connecting front end with back end and achieving 70% of the designs of the wire-frame.
   2. Creating return buttons for all windows except the first window upon opening the app
2. Back end:
   1. Reviewing the code and cleaning it up  
      Solving unwanted exceptions and bringing the back end product to 90% of the desired product.

## Problems that arose during the last milestone phase

1. We encountered difficulties in gathering data across various web pages, primarily due to our limited knowledge of the correct format for writing query parameters in URLs. Our code consistently succeeded in accessing information from the first page but failed to retrieve data from subsequent pages. This challenge was ultimately overcome by relying on intuition and a deeper examination of the rules governing endpoints, as well as a more comprehensive understanding of the conventions for query parameters and variables within URLs.
2. We had programmed buttons in our graphical user interface (GUIs) to facilitate the transition between different windows. However, despite our efforts, the buttons did not function as intended. Although the code was written in accordance with our existing knowledge and seemed correct, the buttons remained unresponsive. This problem was solved by setting the requested window as “visible”, when the button is pressed.
3. We had conflicts in merging Files that was solved manually. Furthermore, we had to grow accustomed to GitHub itself, as most of us had little to no experience with it. Consequently a lot of problems arised when using GitHub’s various commands and features.

## Milestone 2

## Things that have been achieved since the last submission

1. Back end:
   1. All the drone data can now be fetched including the drone dynamics.
   2. The drone dynamics are stored as individual objects in a list and sorted by recency. This is realized by implementing a time class which splits the time string into separate strings which are saved as ‘years, ‘months’, ‘days’, etc.
2. Front-end:
   1. Successful connection of Front-end and Back-end
   2. Implemented a loading screen.
   3. Implemented a first window with a greeting and buttons.
   4. Implemented a following window in which the user can choose between viewing slow, medium, or fast drones by pressing the respective button.
   5. Implemented a window that tabularly displays drones of the same speed class (sorted by their maximum speed) and their basic attributes. Additional information about a drone can be found in a new window which the user can open by pressing an ‘Info’-button.
3. General:
   1. By including the use of threads in the ‘test’-class in which we run the program,  
      new data is being fetched from the server every 5 minutes while running the program.

## Things we are planning to achieve until the final submission

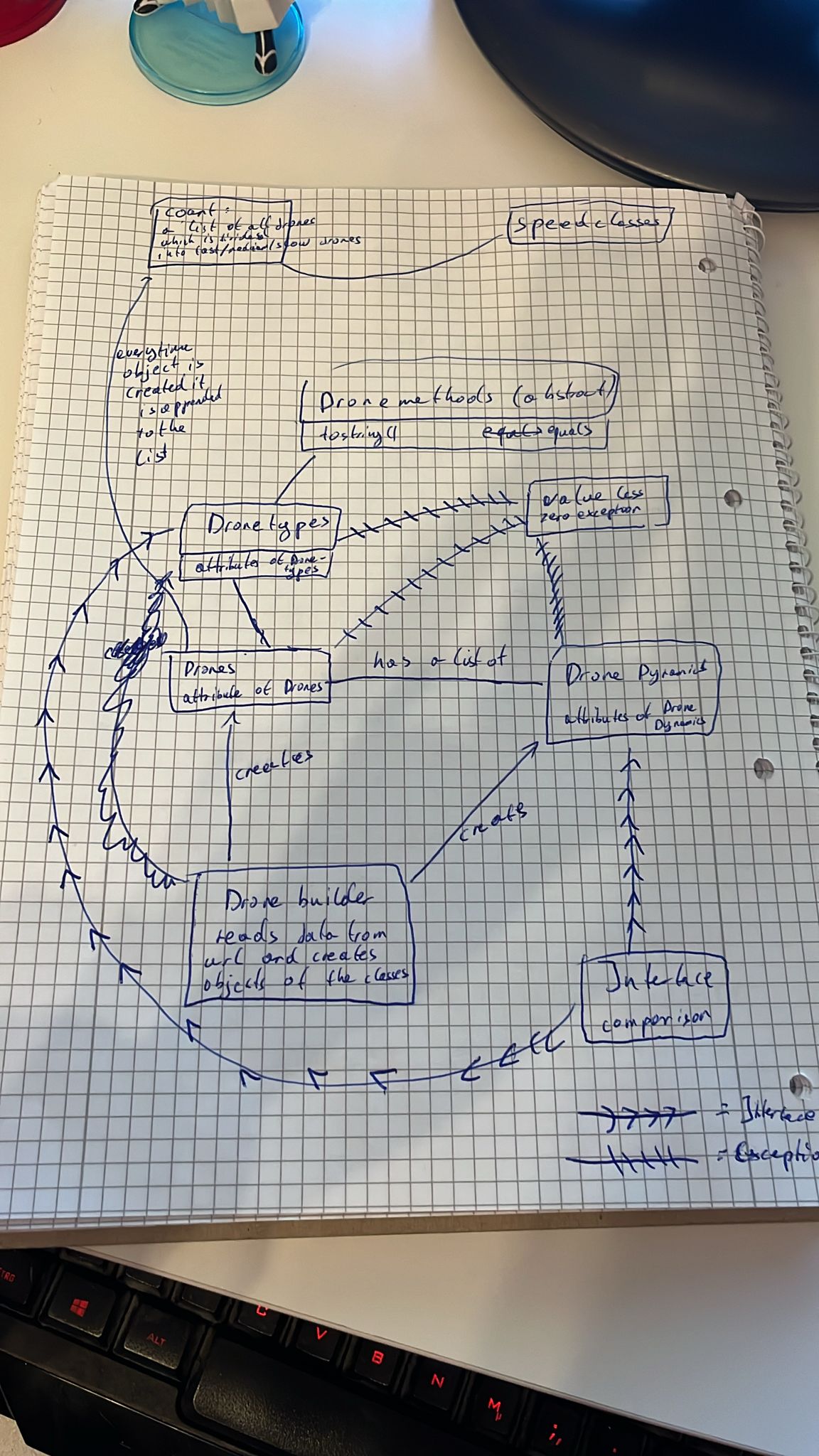
1. Front end:
   1. Fixing a bug which causes the window in which you choose the speed class to reopen twice when pressing the info or back button.
   2. Fixing a bug which causes the first window to be blank unless maximized.
   3. Implement a ‘history’-function which allows the user to search through older drone dynamics data with a search bar or table.
   4. Refresh button to update the data in the ‘speed class’ windows and ‘Additional info’ windows.
2. Back end:
   1. Reviewing the code and cleaning it up
   2. Potentially rewriting some parts to improve runtime and efficiency.

## Problems that arose during the last milestone phase

1. We encountered a bug which causes the window in which you choose the speed classes to be reopened twice when pressing the ‘Back’-button or the ‘Info’-button.
2. If the user were to quickly click through all the windows to view a drone speed class, the program would not be able to fetch the drone data quickly enough. This leads to a blank window until refreshed.   
   🡪 Solution: Loading screen on startup which closes when the program finished fetching the data

\*\*Below are all the referenced diagrams used for development\*\*

# Figures



**A piece of paper with writing on it

Description automatically generated**

Figure 1Initial sketch for back end

Figure 2 Initial sketch of class diagrams

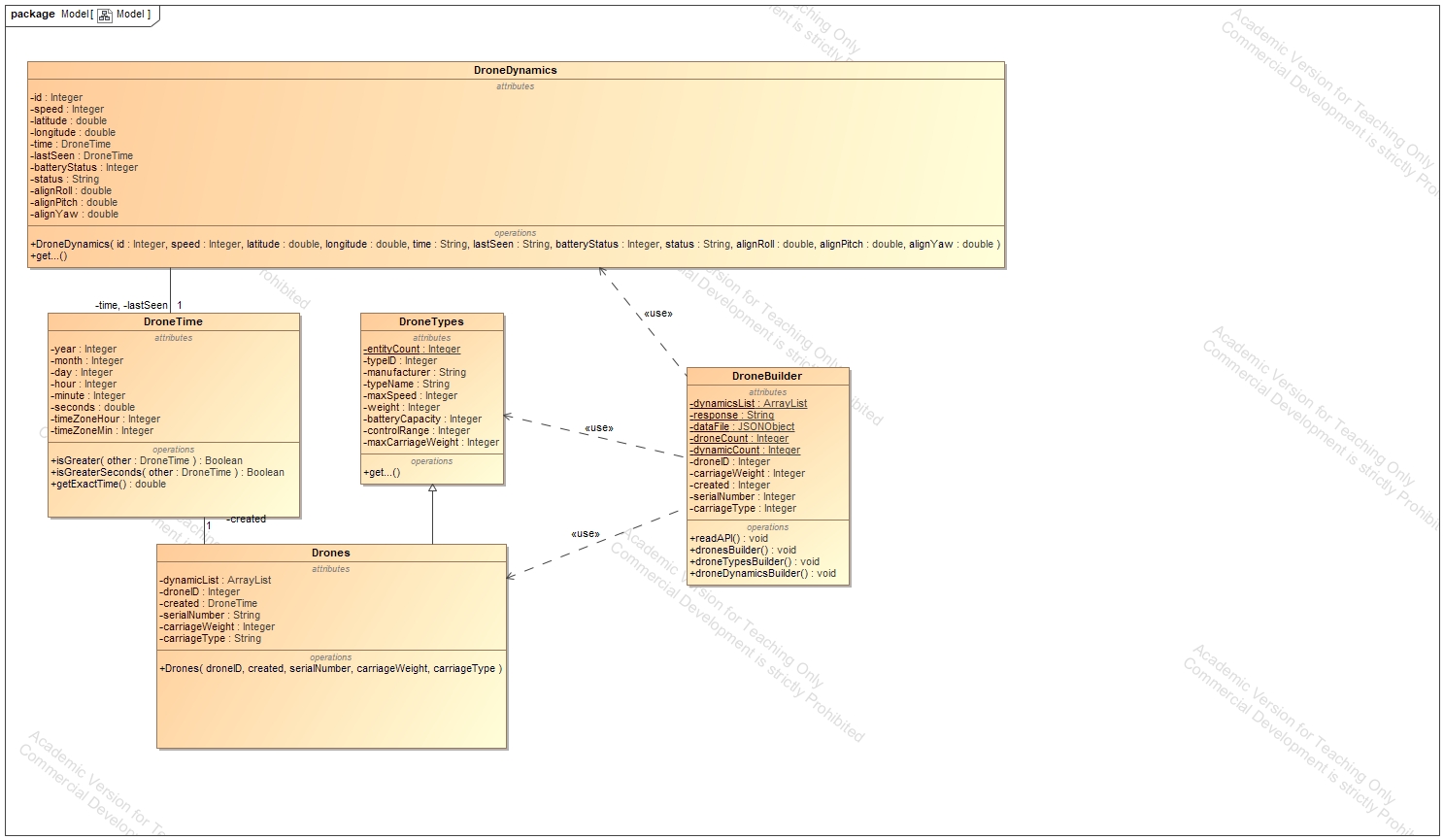
****

Figure 3 Class Diagram done in MagicDraw

|  |  |
| --- | --- |
| Figure 4 UI Prototype | |
| Ein Bild, das Text, Screenshot, Schrift, Reihe enthält.  Automatisch generierte Beschreibung  Figure 5 UI Prototype |  |

A screenshot of a computer screen

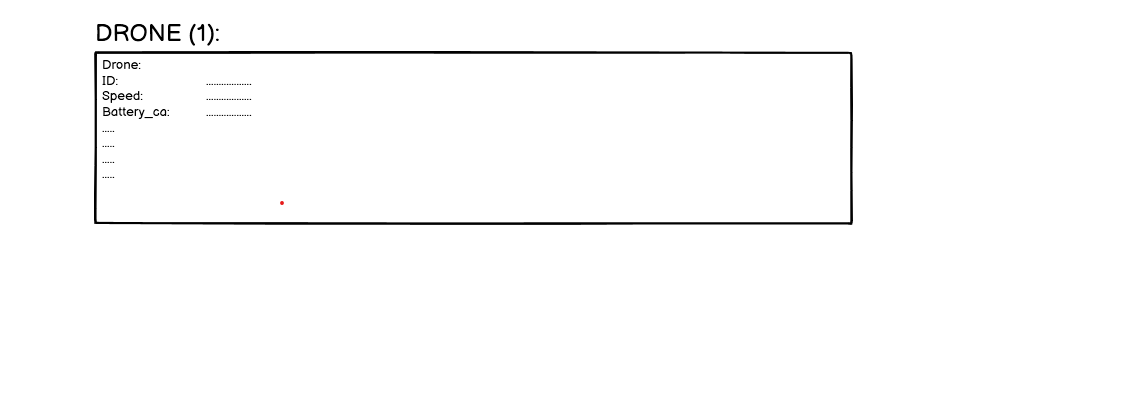
Description automatically generated

Figure 6 UI Prototype

A table of numbers with a number in the middle

Description automatically generated with medium confidence

Figure 7 UI Prototype

Figure 8 UI Prototype

# Technical description

In this project’s scope we were provided with an API containing all the relevant drone data which was divided into three sections - a section for drone types, a section for drones and a section for drone dynamics which correspond to their respective classes in our project.

Everyone drone also has a drone type, therefore, the “Drones” class extends the “Dronetypes” class. An abstract class “DroneMethods” is used to override certain methods like toString() or equals() for our different classes. The last section of the API is a list of drone dynamics. Initially it was thought best to create a drone dynamics class, which extends our drone class. But this idea was scratched due to the sheer amount of drone dynamics.

A solution we found was to implement a “has a” relationship. Essentially every drone has a set of “DroneDynamics” objects stored in an ArrayList.  
Furthermore, we decided to include two exception classes which we deemed were necessary for our project: having certain values below zero (for instance speed) and returning an empty list.

Time based attributes found in the API like “created” for example are stored as attributes of their respective classes of the type “DroneTime” – a separate class which we wrote.  
The “DroneTime” class splits the read string into multiple attributes like “years”, “months”, “days”, etc. and uses methods like getExactTime() to return the time string as a value of the total amount of seconds.  
This is done so we can later sort dynamics by their timestamps.

Fetching the data and formatting it to our needs in separate classes is only half the story, though. Our biggest and most important class is the “DroneBuilder” class. It receives all data from the API and calls the constructor of the drone class (drone types are created via the super constructor), as well as the constructor of the drone dynamics class (this is realised via two threads for an increased speed). Every time the constructor of the drone class is called the created object is appended to a static list of all drones. All drone dynamics are at first also appended to a general list. Afterwards the list of drones is split into three separate lists. The distribution of the drones is based on their maximum speed.  
Furthermore, before these lists are created every drone has its respective dynamics stored in a list. This is realised by every drone parsing through the general drone dynamics list, and appending each object with a corresponding ID, to its own drone dynamics list.

The front end contains multiple classes each representing a window. One class is only for displaying the fact: “loading the data” for instance. While other classes have buttons with actions attatched to them. The task of the front end is to visually display the data of the back end for the user. We will not provide a detailed decription of the many methods arranging the frames or listening to certain actions. One of the only worth mentioning aspect is the attaining of the respective class which is accessed. This is realised partially in the back end. Every time one of the three lists is being accessed a static name attribute is set to the “getter” method’s name.  
This is done so that the program knows which window for the correct drone list to reopen, upon the user pressing the back button.

Additionally, the history window is the idea of the user being able to search the “history” of drone dynamics of a specific drone. The user can type in two time stamps and this will either result in a number of drone dynamics, which were recorded between the two times, or an error box if the difference in time exceeds 10 minutes.

Lastly all of this is brought together in a test class which connects frontend and backend. It also contains a thread which updates the drone data every 10 minutes. (If the user is in a window which already displays data, he/she can update the window to receive the new data.)

# User Handbook

This handbook is a guide to our drone world.  
To start off, start the program either in an IDE by running the test class or by opening the .jar file. A loading screen will appear on the screen. It is important to wait patiently until the program is done loading.

You are presented with a welcome screen which has two buttons asking you if you want to proceed. Both options will forward the user to the actual program and are just meant as a joke. The program will now display the screen in which one can decide which drones he or she wants to see. The choices are between fast, average, and slow drones. After deciding on one, the user is presented with a list of the respective drones’ class. If the user wants to see the other classes, he or she would need to press the back button.

With the refresh button one can manually refresh the data, this could be important as the data presented in the API is dynamic and could change every now and then.

Each drone in this list also has the button “more info”. Clicking this button will open another window which display all other information of this drone. Like the previous window, this window also has the option for the user to manually refresh the data. The window also has a button “history” where the user can search the number of dynamics recorded between two-time stamps, which the user can choose (Note: the difference in time of these time stamps may not exceed 10 minutes).

Opening a window always means automatically closing the previous window, the only exception is the history window which is displayed alongside the “more info” window (Refreshing the “more info” window will also refresh the history window).  
This concludes everything that is important about our program our team hopes that the user can find a suitable way to utilize our program.