TWITTER RECOMMENDATIONS SOFTWARE

Michael Pearson
H446, 15/09/2020 (dd/mm/yyyy)

Contents

Analysis

Introduction

Recommendations are used in many areas, from networking tools to shopping personalisation. Websites such as LinkedIn give networking recommendations to help you expand your connections by suggesting people based on background, geographical location, existing connections, and interests. LinkedIn's implementation is accurate and outputs relevant recommendations that please the user. However, due to the multitude of factors at play, this leaves little room for suggestions outside of your field, and as a result can make the user feeling claustrophobic with the recommendations.

Amazon is a e-retailer and supplies product recommendations to users based on their shopping trends, search trends and alternatives to items in their wish list. Furthermore, Amazon recommends products that are similar but not exact to previously searched/viewed items. As a result, you have a higher chance of expanding your interests by viewing these guesstimate products. While Amazons recommendations are effective and accurate, they can be biased towards higher priced items to increase profits.

The problem

Currently, users may experience lack of interest to their feed if they follow a small number of accounts or accounts with a similar genre. As a result, users will only see a limited number of tweets before running out of content or the content they view will be repeating from many similar accounts reposting. Consequently, users get bored of their feed quickly and become disinterested with the app.

Furthermore, on Twitter, recommendations tailored to the user are limited in accuracy and has room for improvements. Networking recommendations tend to be inaccurate and have no feedback loops. As a result, you end up being shown decreasingly compelling recommendations. This becomes a problem as the user is dissatisfied with the accounts shown and has little control over the output. This results in the user struggling to engage fully and seamlessly into a community/interest.

As of 2021, Twitter has 206 million daily active users[1] and therefore has a large potential for personal network expansion. This leaves a large gap in the market for users to engage in a community that has not yet been fully capitalised.

My proposed solution will fix this by allowing the user to easily expand their social media network by finding new, relevant, and accurate accounts for the user to follow. This leads to the user becoming more engaged with their feed as they will have a wider variety of content to view. In addition, users interested in a certain genre, such as gaming or football, will become more integrated into the respective community as they follow more accounts within that community. This results in the user having an expanded network on the communication app Twitter. I will do this by using a series of algorithms and the twitter API.

[1] Statista.com, July 2021, <https://www.statista.com/statistics/970920/monetizable-daily-active-twitter-users-worldwide/>

Why is the problem suited to a computational solution?

Decomposition

The problem is suited to decomposition as it contains sub-problems, including but not limited to:

- Twitter account handling, including logging in and storing the user's account
- Efficiently recommending other twitter users to follow
- Enhance recommendations based on feedback from the user
- Intuitive graphical user interface

Decomposition is, by definition, breaking down the complex problem into smaller parts. These sub problems can be easier to conceive, program and maintain. This lends itself to computational methods as development time is reduced due to clear milestones that can be individually developed, avoiding the use of waterfall development practice.

These problems can all be divided into even smaller sub problems, such as the different aspects of the GUI, in the module hierarchy chart of the design section.

Abstraction

Abstraction can be used to remove unnecessary details from the problem, such as the colour scheme of the interface (which does not matter until much further along in the design process). This will allow me to focus on the more important problems such as designing an efficient algorithm, while not having to worry about the smaller, less important details. Furthermore, for the end user, the solution is abstract as it is not necessary to know how to code or what the code does in order to use the application.

Parallel processing

Finally, the problem is suited to a computational solution as the program will have to process large sets of data, which lends itself to parallel processing as the data is independent of each other, therefore can be processed in parallel. This will significantly decrease waiting times because modern processors have multiple cores and threads, creating a more proficient user experience. Furthermore, the program will be looking for general trends within a set of data, known as data mining. In my case, the set of data is someone's tweets, and the trends will be the interests that appear within the tweets.

Algorithmic approach

The program may be able to implement a variety of algorithms, such as a depth-first search or a breadth-first search can be used to find new users to follow. Furthermore, a keyword extraction algorithm (an algorithm to extract words from a document or text that define it) can be used to extract keywords from tweets to iterate over to find matching interests.

Justification

Overall, due to the computational methods listed above, such as parallel computing – which can only be achieved with a computer, my application is suited for a computational approach, due to the benefits created for the user. Furthermore, Twitter is an online application, replacing communication methods such as letters, therefore the solution lends itself to a computational solution due to the nature of the problem.

Stakeholders

The demographic for my stakeholders is those who use Twitter. While this is the case, I also want to capture the use cases of some who have never used the platform, to ensure my program also caters for these people. As a result, I have chosen a mix of users, ranging from advanced to never having used twitter.

Stakeholder	Relevancy	Availability	Link to page
Chirag	Occasional twitter user	Weekly	Chirag
Sam	Daily twitter user	Daily	Sam
Emma	Has never used twitter	Daily	Emma
Fin	Advanced twitter user	Weekly	Fin

Two of stakeholders, Fin and Sam, use Twitter every day. Fin works for a media twitter account, and therefore spends a lot of his day using Twitter and posting about news/football stories. Fin will use the solution in order to expand his network on Twitter, allowing him to make new connections within his media company, as well as being recommended other news media accounts. Sam follows many football accounts and therefore will be recommended new football/related sporting accounts to follow. Chirag and Emma are at the other end of the scale. Chirag occasionally uses Twitter whereas Emma has never used Twitter before. Chirag follows Formula 1 and therefore will be recommended accounts regarding F1 and other such fan accounts. Emma does not use twitter, and therefore the solution will be used to find new friends with common interests. The solution is appropriate to these stakeholders as it will help find more people and posts to follow and interact with respectively.

Stakeholder Questions

These are the questions I will be asking each stakeholder. These should help me determine the features of the program and how users would be likely to use/interact with the program.

1. *How often do you run out of content to view on Twitter?*
2. *Do you ever get bored of your Twitter feed?*
3. *What interests/subject areas do you follow on Twitter?*
4. *What features would you expect from a recommendation service?*
5. *Are there any additional features you would like to suggest?*

Stakeholder Interview Answers

When interviewed, all stakeholders preferred to use a GUI to interact with the program (instead of alternatives such as command line interface). Furthermore, I asked each person to draw a sketch of how they would like the program to be laid out. This is useful for me as it will allow me to design the GUI structure easier, as well as taking in stakeholder considerations to produce a concept that they all like.

Chirag

1. After a few minutes of scrolling through my feed, I find myself catching up with where I left off from.
2. Yes, my feed does become boring after seeing the same content reposted by many similar accounts.
3. I follow Formula 1 content, as well as aviation.
4. Recommends me relevant new users to follow that I would be interested in and searching for specific interests
5. Dark/light themed GUI option

Sam

1. I do occasionally run out of content on my feed; however, I follow hundreds of accounts to do with football and so only rarely.
2. After scrolling through my feed for a while, I get bored as a lot of the content is the same football content. Sometimes I wish Twitter would show my slightly different content I may be interested in.
3. Football and gaming such as Rainbow Six.
4. Finds users that like similar things as me as well as showing me new, slightly different content that might interest me.
5. Recommendations with feedback options (like/dislike) on twitter accounts to change future recommendations, follow button, block button

Emma

1. Currently I only follow my friends on Twitter, who do not tweet much, and therefore I run out of content quickly.
2. I use twitter very rarely, but when I do I find myself getting bored quickly due to the lack of accounts that I follow.
3. I only follow my friends currently.
4. Finds me new relevant accounts to follow.
5. Search bar for certain interests.

Fin

1. I follow thousands of news and football accounts on Twitter and therefore have a constant stream of tweets on my feed.
2. I don't find my Twitter feed interesting as I spend so much time looking at it. I don't follow genres I enjoy as these are not relevant to my media job. So yes, I am bored of my feed often.
3. News/football accounts
4. Allowing me to search for a genre of accounts that I find interesting. The window should be vertical rather than horizontal. Clicking on an account should open it in Twitter
5. Showing metrics about a tweet such as number of likes, retweets, etc.

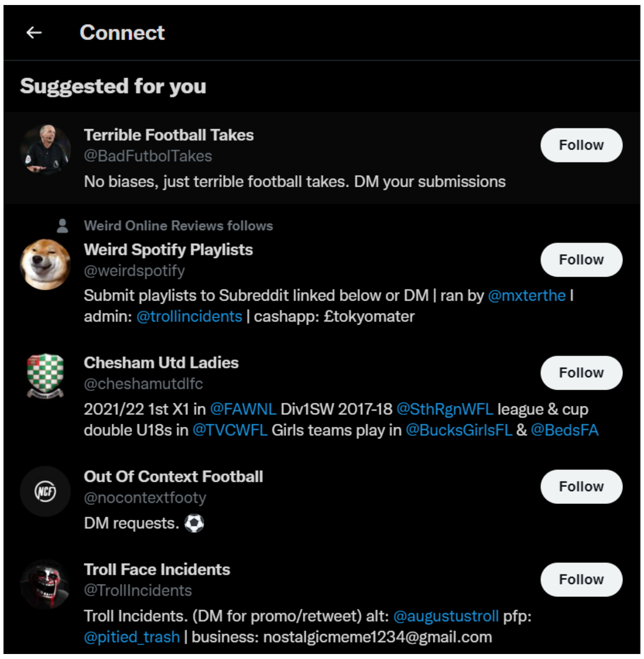
Research and data analysis

From my stakeholders' answers, the key features that I will add are recommending relevant results only, as 3 of 4 of my stakeholders requested this feature. This will ensure that the accounts they are recommended are related to their current interests as reflected by their current following, this will allow the user to expand their network within a certain community. Furthermore, Chirag and Fin suggested that they should be able to search for a certain interest/genre and receive recommendations based on that search. This will allow the user to expand their network into new fields that they may not currently be following in. This allows a user who is discovering a new hobby, for example, to dive deeper into this interest.

Similar solutions

- "Scraping experts: Twitter scraper"
  - Has 3 payment options: free trial, monthly ($19 USD), annually ($169 USD)
  - Important data ("#tweets, Followers, Following Name, Profile, Source Link")
  - GUI ("One screen dashboard")
  - Exporting ("Export Scraped Data in csv file")

- Twitter's "Who to follow" widget
  - Shows a vertical list of twitter accounts, recommended by 'Suggested for you'
  - Details shown are profile picture, username, twitter handle (@name), biography (aka bio), follow button
  - Links between accounts, e.g., Account 2 (followed by user you follow)
  - From this design, I can apply parts to my program. I find the layout easy to use, which is a success criterion for my design. This is important as a simple design will allow a wider range of people to use the program, not just advanced users.



Terrible Football Takes
@BadFutbolTakes
No biases, just terrible football takes. DM your submissions

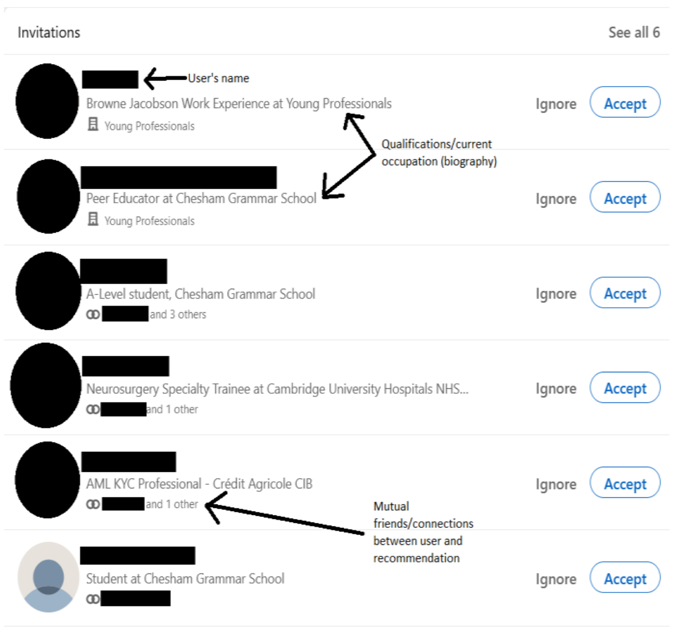
Weird Online Reviews follows
Weird Spotify Playlists
@weirdspotify
Submit playlists to Subreddit linked below or DM | ran by @mxterthe | admin: @trollincidents | cashapp: £tokyomater

Chesham Utd Ladies
@cheshamutdlfc
2021/22 1st XI in @FAWNL Div1SW 2017-18 @SthRgnWFL league & cup double U18s in @TVCWFL Girls teams play in @BucksGirlsFL & @BedsFA

Out Of Context Football
@nocontextfooty
DM requests. ⚽

Troll Face Incidents
@TrollIncidents
Troll Incidents. (DM for promo/retweet) alt: @augustustroll pfp: @pitied_trash | business: nostalgicmeme1234@gmail.com

- LinkedIn's networking tools
  - Displays user's name and uploaded profile picture
  - Qualification/occupancy below the user's name
  - Mutual connections are emphasised by a chain link symbol
  - For each account, the user may ignore/block the request or accept/follow the user
  - From LinkedIn's design, I will use the mutual connections as this makes the user trust the recommendation more. Furthermore, the button to ignore the user is a good idea and I will use this concept to allow users to block certain accounts that they do not want to see again.


User's name
Qualifications/current occupation (biography)
Mutual friends/connections between user and recommendation

Invitations See all 6

[redacted] [redacted] Browne Jacobson Work Experience at Young Professionals
Young Professionals
Ignore Accept

[redacted] [redacted] Peer Educator at Chesham Grammar School
Young Professionals
Ignore Accept

[redacted] [redacted] A-Level student, Chesham Grammar School
[redacted] and 3 others
Ignore Accept

[redacted] [redacted] Neurosurgery Specialty Trainee at Cambridge University Hospitals NHS...
[redacted] and 1 other
Ignore Accept

[redacted] [redacted] AML KYC Professional - Crédit Agricole CIB
[redacted] and 1 other
Ignore Accept

[redacted] Student at Chesham Grammar School
[redacted]
Ignore Accept

Proposed Solution

When my program is initially opened, if this is the user's first time using my application, they will be presented with a login page, as they need to authenticate with their Twitter account before they can use the features of my program. Therefore, it is essential to have a login page first.

My program will be a GUI with two separate main areas, the tool bar at the top and a larger area below containing the main content of the current screen. The user will be presented with a vertical list of accounts to follow, as inspired by Twitter and LinkedIn's solutions. Upon clicking on an account, an expanse will appear to give more details, such as recent tweets and options to open in twitter, follow or block this account.

Essential features

After my research, initial conceptualisations, and stakeholder interviews; these are the essential features I propose.

Feature	**Explanation**
Login GUI	The user must authenticate with Twitter before my program can use the API. If they are not authenticated, my program will not work. Therefore, they must be able to login to their Twitter account first.
Tool bar	Contains basic information, such as the Twitter account they are logged in with, and a button to go to the settings page
Recommend-ations	Based on similar interests, this is the main part of my program. I will use keyword extraction algorithm to determine interests.

Vertical list of recommend-dations	Oriented in a vertical manor, with text boxes stacked on top of each other (similar to this word table graphic!). This is because text reads left to right and therefore has more readability and space.
Follow button	Follows the recommended account to save the user time, as otherwise they would have to open Twitter, search the user, and then follow them.
Ignore button	Blocks the account from being recommended again, so if the user does not want to see a particular account again – for whatever reason – they can block the account.
Mutual connections	For each recommendation, any mutual connections you have with the other account will be displayed, as per LinkedIn's design.

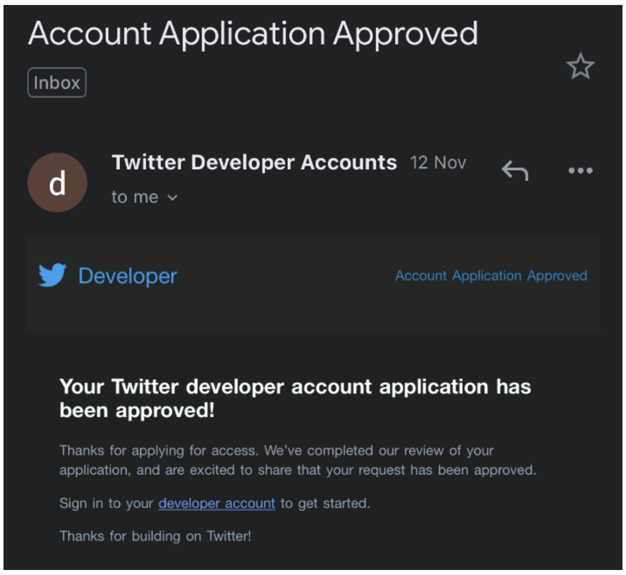
Limitations

One limitation behind my solution is that there are a few different variations of the Keyword Extraction Algorithm – the main algorithm behind my recommendations. Each variation is suited to a slightly different scenario, and therefore I may have to code and test a few different variations before I find one that is particularly suited towards my application. This means development could take longer, and time constraints could become a problem. If time constraints do become a problem, I will have to test the variations I have coded at the time and determine which is the best for my solution – however it may not be perfect.

Furthermore, I am limited to features the twitter API offers, with the access permissions that are granted by twitter, however this should not be too delimiting as everything I need to do in my program should be possible by the API. In addition, to process tweets I need to retrieve a large amount of them using the twitter API, which could be subject to rate limitation by twitter. This means that I would only be able to access a certain number of tweets at a time which could make my program appear as if its slower. To solve this, I would have to buffer account recommendations, retrieving and processing X number of tweets at a time.

Furthermore, using any libraries such as the java GUI library could pose limitations as the code may not be as flexible as I need and could be subject to time limitations as I would first have to take the time to learn the library in depth to understand how it works before I am able to start using it in my application.

Finally, to develop my program, I need access to the Twitter API. I had to apply directly to twitter for approval for my program to use the API. I had to answer many questions about what my project would do and how it would use the twitter API.
Here shows the email I received from Twitter approving my use case for their API.


Account Application Approved

Inbox

Twitter Developer Accounts 12 Nov
to me

Developer Account Application Approved

Your Twitter developer account application has been approved!

Thanks for applying for access. We've completed our review of your application, and are excited to share that your request has been approved.

Sign in to your developer account to get started.

Thanks for building on Twitter!

Confirmation of concept

After my initial concept planning, I sent an email to my stakeholders proposing my concept, to get feedback and approval on my design to begin further design. The email I sent was as follows:

Dear Stakeholder,

I am sending this email today to inform you of and seek feedback on my initial concept of design for my program 'Twitter Recommendations'. After researching similar programs, I have incorporated all stakeholder designs as well as aspects from my research to conclude on an initial concept.

As with most programs, a graphical user interface will be used to create a better user experience. A vertical list of twitter accounts will be displayed based on your interests. Upon clicking on an account, a dropdown expansion box will open giving more details on the user with further options to follow, block, etc.

Please inform me on your impression of this initial concept, as well as any feedback you may suggest.

Yours sincerely, Michael.

Stakeholder responses

The responses I received from my stakeholders went as followed:
Chirag: "I like the idea of the GUI with the drop-down menu"
Sam: "I like the concept"
Emma: "I approve of this concept as I think it will be easy to use for me"
Fin: "Yes I approve of this concept"

Requirements

Software requirements

As I am coding my program in Java, the end user will need to have Java Runtime Environment (JRE) installed. This is necessary for my program to run. However, most computers have Java installed (estimated at 13 billion devices running Java), and if not, it is very easy to install. Java is compatible with all operating systems (Windows, macOS, Linux), due to its compiled and interpreted nature, making use of the JRE to run natively on these platforms.

My application makes use of libraries (such as the Twitter API library – Twitter4J, and Java's GUI library) however these will come packaged with the program files and therefore the user will not need to take any action to install these.

Hardware requirements

My application will have to process large data sets of strings, and so will need the computational power to do so. In addition, I may make use of parallel processing using multiple threads, and therefore a dual/quad core CPU may be required. However, most modern computers have a processor powerful enough for this use case.

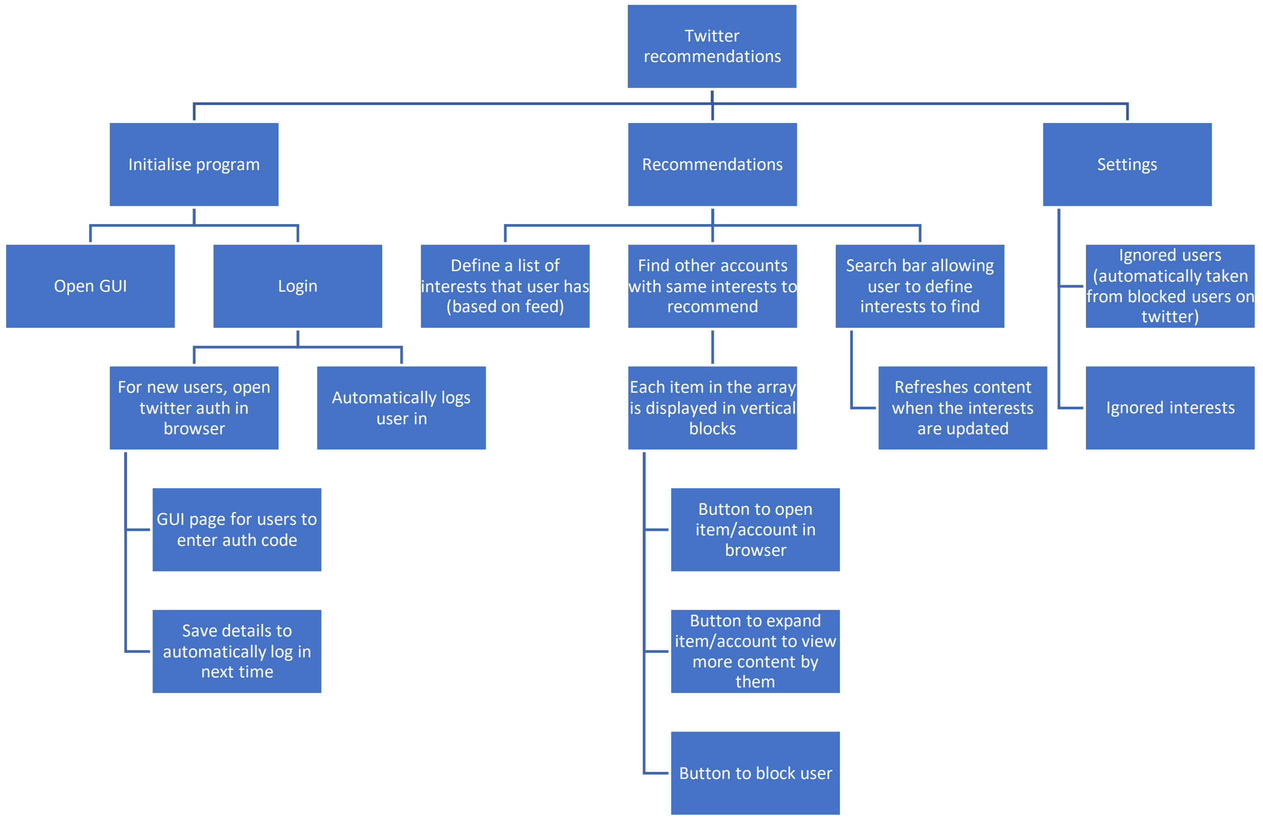
Requirement Number	Requirement	Justification	Success Criteria & Evidence
1. Login page/module			
1.1	Login UI is only shown to user if they are not logged in or it is the first-time using application	The user only needs to authenticate with Twitter once, and therefore this page only needs to be shown to the user once	Login UI is displayed first-time, but not again once they have authenticated. Screenshots of the menu taken to once loaded with and without login data.
1.2	Button to open authentication in default browser	Using twitters website, the user must get an authentication code. Furthermore, using the default browser the user feels safe as this is their choice in browser	Clicking button opens users default browser to Twitter's authentication website. Screenshot of correct website
1.3	Number pin field (buttons)	Allows user to enter the authentication code provided by Twitter	Screenshot of buttons that user can press + their effect once pressed
1.4	Logs in once valid authentication code is provided	The code will be used to log in with Twitters API and the rest of the application can then be used. The user will be taken to the main screen	Entering correct code changes UI screen to main recommendations. Screenshot of before + after entering correct code
1.5	Authentication data is saved to file	This is required so that the user doesn't need	After the user has authenticated for the first

		to authenticate each time the load the application	time, the data regarding their account is stored to a file on their computer. Screenshot of stored data
2. Recommendation module			
2.1	Rectangular boxes stacked vertically	Text is read left-to-right and therefore is more readable	Screenshot of correct layout
2.2	Boxes expand upon being clicked	To give more information on a particular account, the box will expand once clicked to display this	Box must show more information when the user clicks it. Screenshot of expanded information view
2.3	Boxes display information about recommendations	Information such the profile picture, account name, etc will allow the user to decide whether to follow	Screenshots of information being displayed
2.4	List of interests based on user	To give recommend-dations, a list of interests is needed about the account, which is gained via the Keyword Extraction Algorithm	The algorithm is given a text and will return a list of strings that are the important parts of the text. Screenshots of the input and output texts
2.5	Follow button	Saves time for the user as they can quickly follow the account from the program	Once the button is pressed, the user will now be following the account on Twitter. Screenshots of before vs after button press
2.6	Ignore button	Allows the user to block the account if they do not want to see it again	Once the account is blocked, they will never be recommended to the user again
2.7	Interest search bar	Allows the user to search for a particular subject/topic they are interested in	Once searched, the user will receive recommend-dations based on that interest. Screenshots of relevant accounts being displayed
2.8	Stop words	Stop words are words that will be excluded by the algorithm and so will save processing time as non-important words will not be processed	Algorithm does not output any stop words with pictures to demonstrate this working

2.9	Match languages	Only processes texts/tweets that are English (including US English) to match the stakeholder’s language and reduce inaccurate recommendations	Algorithm should exclude processing texts from users that are not English speaking
2.10	Refresh button	A button that will generate a new list of recommendations for the user	Upon pressing the refresh button, the user should have feedback that their input did something while new recommendations are generated. Screenshots of this should be provided

Design

Module Hierarchy Chart


Twitter recommendations
Initialise program
Recommendations
Settings
Open GUI
Login
Define a list of interests that user has (based on feed)
Find other accounts with same interests to recommend
Search bar allowing user to define interests to find
Ignored users (automatically taken from blocked users on twitter)
For new users, open twitter auth in browser
Automatically logs user in
Each item in the array is displayed in vertical blocks
Refreshes content when the interests are updated
Ignored interests
GUI page for users to enter auth code
Save details to automatically log in next time
Button to open item/account in browser
Button to expand item/account to view more content by them
Button to block user

Explanation and Justification

The above is the module hierarchy chart for my project and is broken down into the three main GUI pages: login, recommendations, and settings. I have decomposed my project in this way as these three pages are independent of each other. Each page is then decomposed further into its main functions.

The login module is important as it will allow my program to use the Twitter API to access the user's feed, following and other such things that would otherwise not be accessible. To securely do this, an authentication token and secret is required to login. For new users, this must be gained once and can be stored to a file to login again.

The recommendations module is the main hub of my program and will be where the user spends most of their time and interacts with the most. In this module, first a list of interests must be defined. This will be created by searching the users feed and using a keyword extraction algorithm to determine interests. Once the program has a list of interests, it can recommend accounts to the user to follow.

The settings module is the smallest module but will still be important. It will allow the user to block accounts/interests from appearing in their recommendations. This is a quality-of-life feature but will ensure that the user is happy and in control of what content is displayed to them.

Individual functions within these modules are then further decomposed to show the exact steps that must be taken to successfully complete the function. For example, the individual buttons that will appear for each recommendation and their purpose.

I used a hierarchy chart as it allows me to easily break each part of my solution down into very simple sub problems that can each be solved independently.

Structure

Problems	**Definition**
Open GUI	In this step, I will open the window to be displayed to the user. For this the Java graphics library (AWT – Abstract Window Toolkit) will be used. The window will be set to a preset size, centered on the user's screen, and displayed on screen.
Open Twitter auth in browser	In the login page, a button will be present that when pressed, will open Twitters authentication website in the user's default browser.
Page for users to enter auth code	Either a design like Apple's passcode with 10 circles ranging from 0-9 or a text box that the user can enter the digits into. Once all digits are entered (Twitter's auth codes are 6 long), it will automatically be processed.
Save details	If authentication is successful, Twitter's API will provide a user token and secret. These are two strings that can be saved to a file. When logging in a second time, these two keys can be supplied to the API to automatically authenticate with the users account, so the user does not need to go through the hassle of the authentication process again.
Automatically log user in	If the user has successfully authenticated before, and the API keys are saved to a file on the user's local machine, these can be

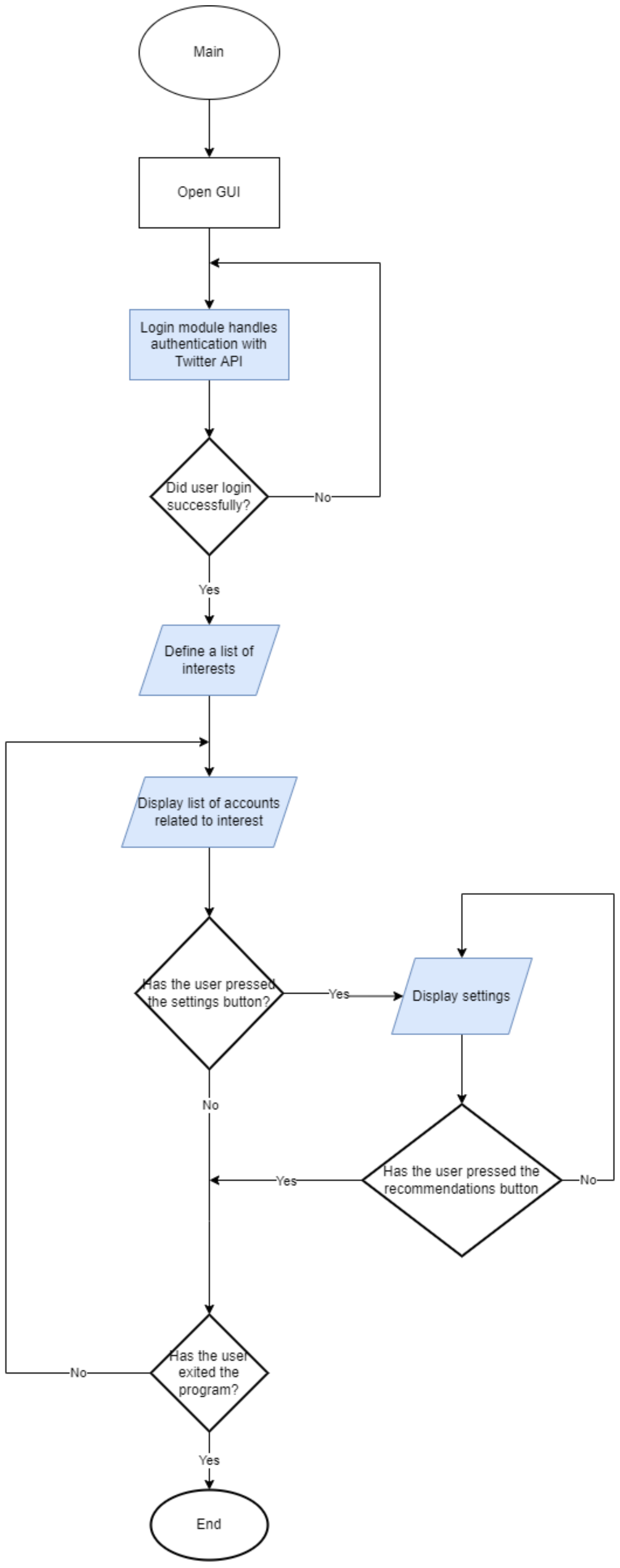
	retrieved on program start and supplied to the API to login without the need to authenticate again.
Define a list of interests	The user's feed is searched and for each tweet, a key word extraction algorithm is applied to retrieve the focus/interest of the tweet. This is a process that is continuous in the background, using a different processor/thread and builds up an increasing list of interests over time.
Find accounts with same interests	A breadth-first/depth-first search can be used to find new accounts, a list of interests of this account are then established to determine whether they are relevant to the user's interests. If it is a match, the account is recommended to the user.
Each account is displayed vertically	As taken from the similar solutions, such as LinkedIn's, each account will appear vertically stacked above one-another.
Button to open in Twitter	For each recommendation, a button can be pressed that will open the account in Twitter on the user's default browser.
Button to expand	When pressed, this box will expand to display more content from this account and will allow the user to gain a larger understanding into the account if they are on the fence.
Button to block user	When pressed, the account will be added to the blocked list and this account will never be recommended again. The blocked users list can be viewed and modified in the settings GUI.
Search bar for interests	User can search for a specific interest, or multiple. My program will then find accounts that match these interests to recommend to the user.
Ignored/blocked accounts in settings	Accounts that the user has blocked on Twitter will be automatically imported and added to the block list in my program. Furthermore, when the user blocks an account in my application, they will be added to this list. From here, the user can view and unblock accounts.
Ignored interests	The user can add keywords to this list which will then be less likely to appear in recommendations.

Algorithms

Main flowchart

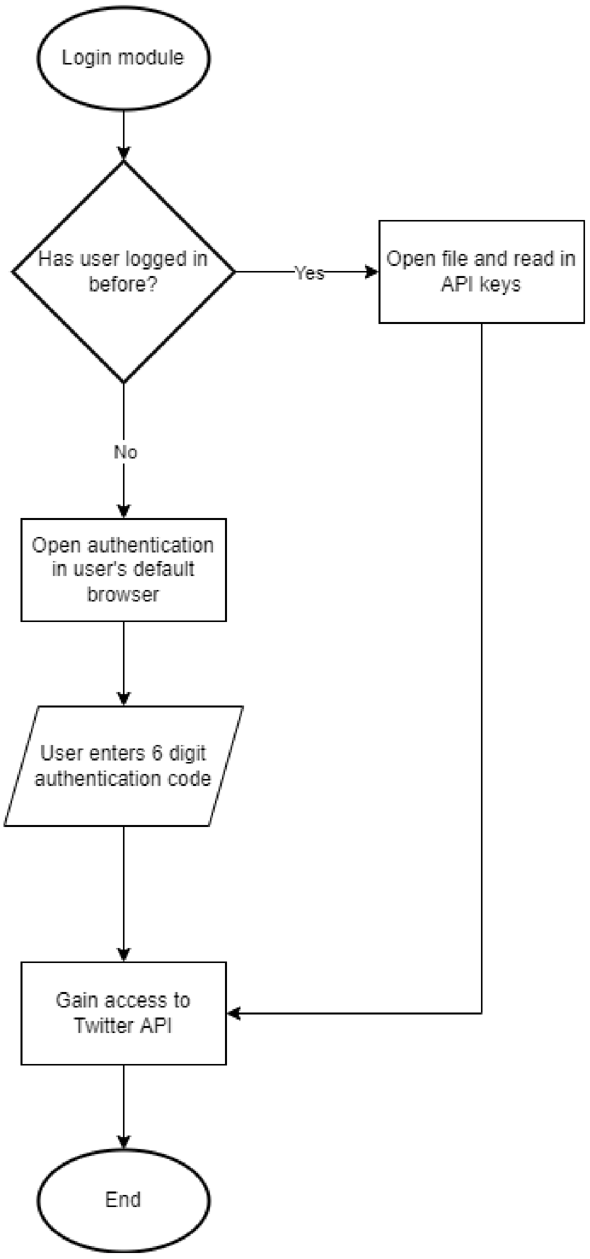
This flowchart describes the overarching process of my program in an abstract and decomposed way. Each module that is decomposed further appears in blue and will have its own flowchart to describe it in more detail. As the flowchart to the right shows, the individual modules in blue come together to create a whole program because there are a series of steps that link each module into the program. Any process that is not coloured is a simple step that will be at most a singular function and thus does not need its own flowchart.

This flowchart shows main loop of my program, from opening the program to the user closing the window.


Main
Open GUI
Login module handles authentication with Twitter API
Did user login successfully?
No
Yes
Define a list of interests
Display list of accounts related to interest
Has the user pressed the settings button?
Yes
Display settings
No
Has the user pressed the recommendations button
Yes
No
Has the user exited the program?
No
Yes
End

Login module flowchart

This flowchart describes how my program will authorise the user and use the Twitter API to gain the appropriate access that my program requires to function properly.


Login module
Has user logged in before?
Yes
Open file and read in API keys
No
Open authentication in user's default browser
User enters 6 digit authentication code
Gain access to Twitter API
End

If the authentication process is successful, the API will return two strings to my program, a token and secret. These are unique to both the user and my program and can be used to automatically authenticate again with the API. As my program is intended to be used on a user's local machine and all data is stored on their machine, I do not feel the need, at this time, to encrypt these strings.

If the user has already logged in before, they will not need to authenticate my application with Twitter as the authenticating token and secret will be saved to a file, therefore they can be read in and submitted to the API to authenticate.

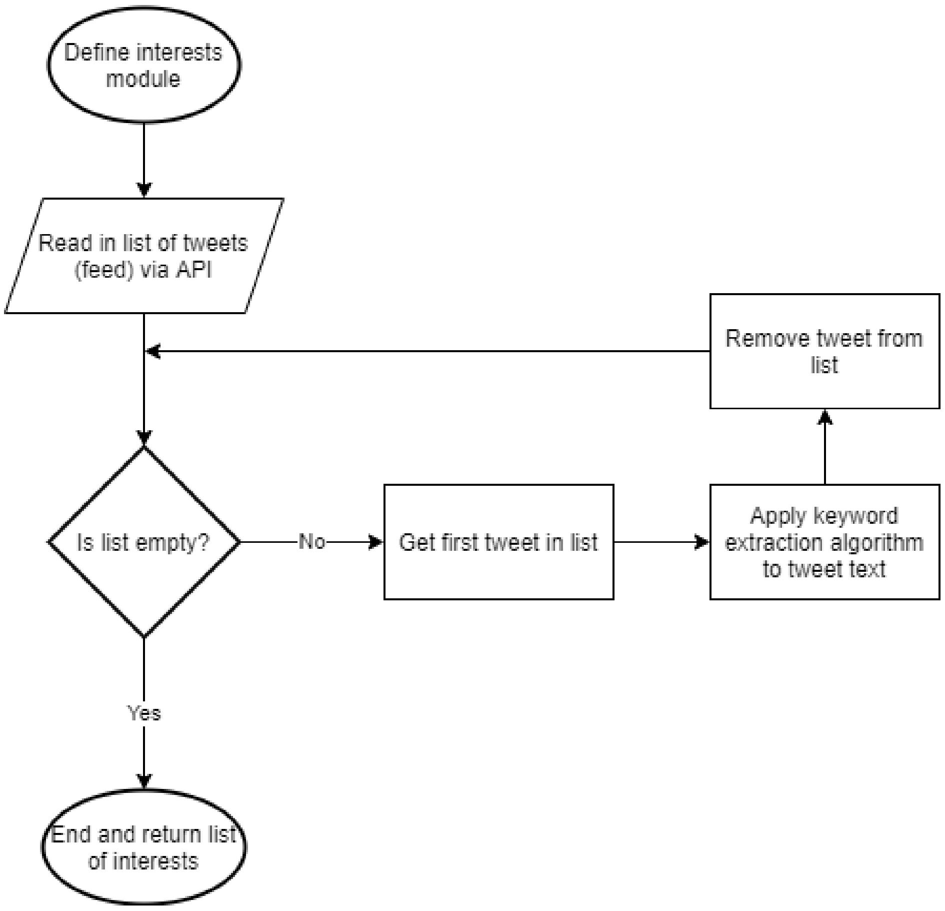
From my requirements, this flowchart outlines the login module and requirement 1.X from my requirements section of my analysis.

Interest's flowchart

The flowchart below shows how the interests of the user will be defined, using a keyword extraction algorithm (which is detailed later in this section). These interests will then be the key words that are used to search for new accounts with similar interests.

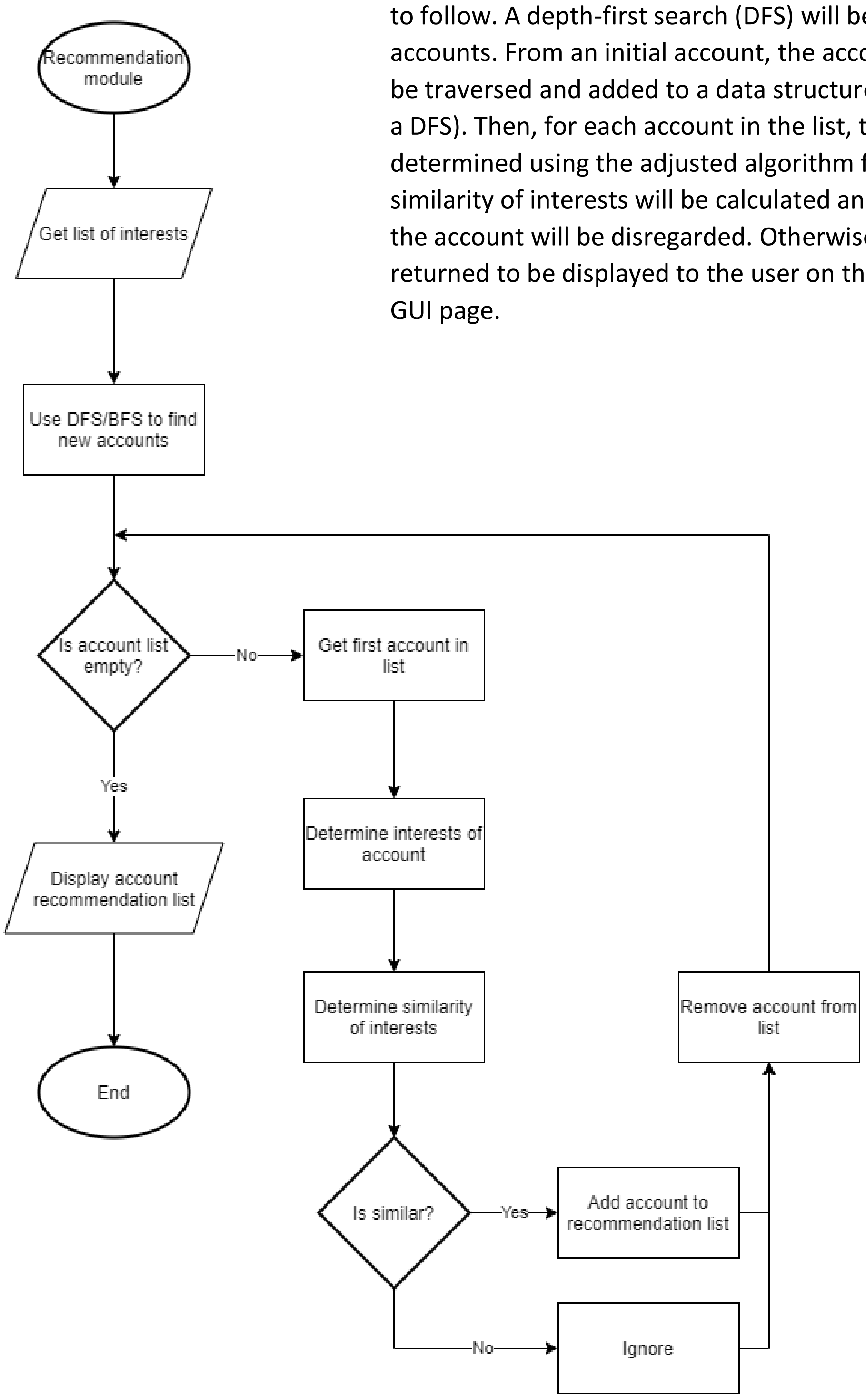
This function can be adjusted to also determine the interests of other accounts based on their posts (rather than reading in feed, adjust to read in posts).

From my requirements, this flowchart outlines requirement 2.X.

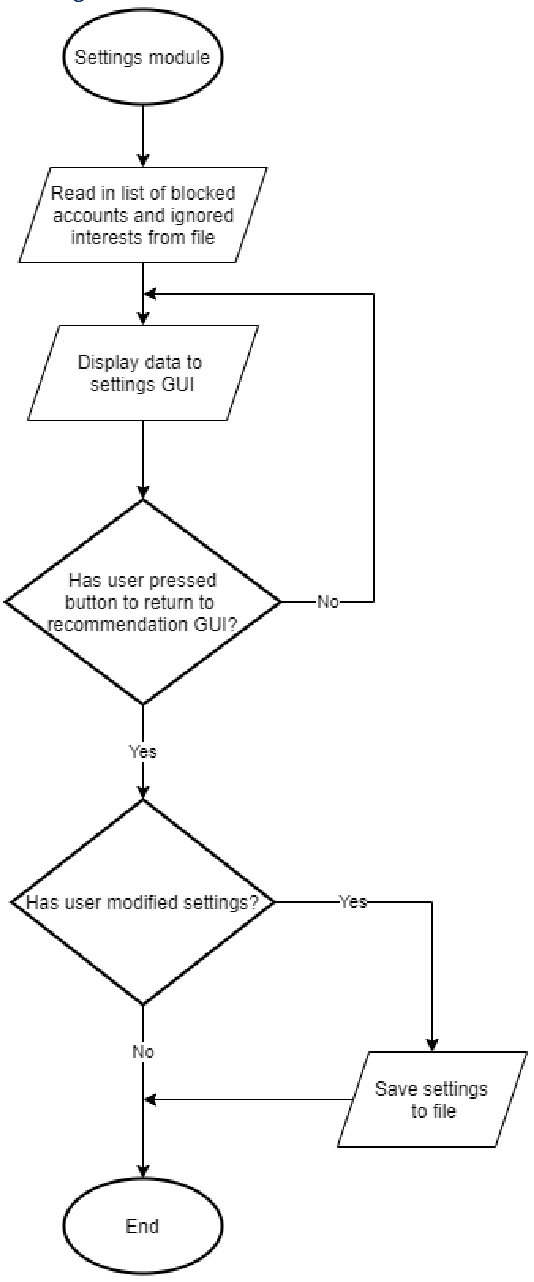

Define interests module
Read in list of tweets (feed) via API
Remove tweet from list
Is list empty?
No
Get first tweet in list
Apply keyword extraction algorithm to tweet text
Yes
End and return list of interests

Recommendation flowchart

This flowchart describes how this module will find the accounts to follow. A depth-first search (DFS) will be used to find accounts. From an initial account, the accounts they follow will be traversed and added to a data structure (such as a queue for a DFS). Then, for each account in the list, their interests will be determined using the adjusted algorithm from above. The similarity of interests will be calculated and if below a threshold, the account will be disregarded. Otherwise, the account will be returned to be displayed to the user on the recommendation GUI page.


Recommendation module
Get list of interests
Use DFS/BFS to find new accounts
Is account list empty?
No
Get first account in list
Yes
Display account recommendation list
Determine interests of account
Determine similarity of interests
Remove account from list
End
Is similar?
Yes
Add account to recommendation list
No
Ignore

Settings flowchart


Settings module
Read in list of blocked accounts and ignored interests from file
Display data to settings GUI
Has user pressed button to return to recommendation GUI?
No
Yes
Has user modified settings?
Yes
No
Save settings to file
End

This flowchart shows how the settings GUI module will function. This will be a separate GUI from the main recommendations page, and so requires a back button. Upon exiting this GUI, any modified settings are saved to file.

From my requirements, this meets requirement 3.X.

Summary

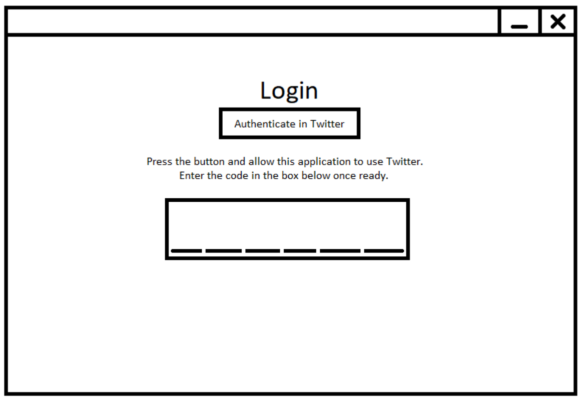
These flowcharts form a complete solution to my problem as they meet all of the requirements outlined in my analysis and fit in to my program in a logical order as described in the overall structure flowchart.

Usability features

Feature	Justification
Important information is displayed centrally	Information such as the login buttons and recommendations will be central to the screen. This area will have more space and therefore information does not need to be cramped, improving readability.
Useful buttons on top-left	According to a survey[2], the top-left corner of the screen gets user's attention first. As a result, important buttons such as the button to open the settings GUI will be placed here, as these are important for the user.
Expanding boxes	Recommendations will, at first, be short summaries. However, the user can click on a box to expand its size. As a result, the recommendation will have more area on screen and so can display more information. This means that information does not need to be cramped, again improving readability.
Colour scheme	My program will have two colour schemes, a light and dark mode. Dark mode is beneficial to the user, as it is light text on a dark background. This reduces blue light exposure (which affects your sleep rhythm[3]).
Contrasting colours	In settings, an option will allow the user to choose the colour schemes (if light and dark mode are insufficient). This allows the user to customise the program to their needs and to choose colours that are suitable for their needs.

GUI Layouts

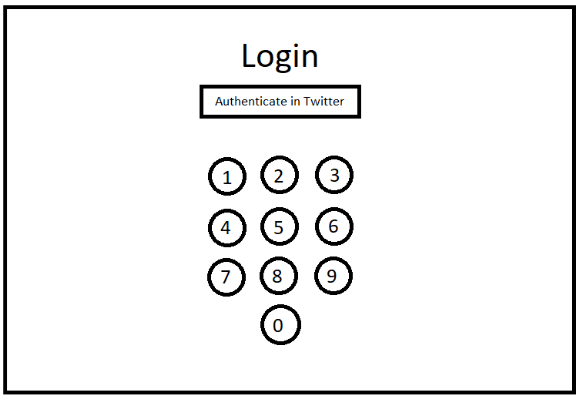
Considering the usability features, similar solutions, and stakeholder input, I designed these mock GUIs for each of my program's main modules.


Login
Authenticate in Twitter
Press the button and allow this application to use Twitter.
Enter the code in the box below once ready.

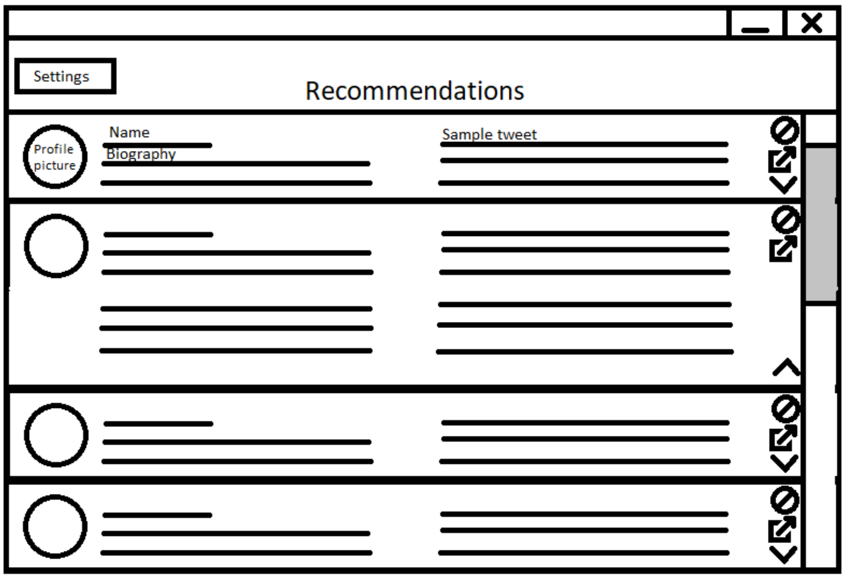
Initially, for my login page I came up with this design. The authentication code would be entered into the box and a button is present to direct the user to authenticate my application in their default browser.

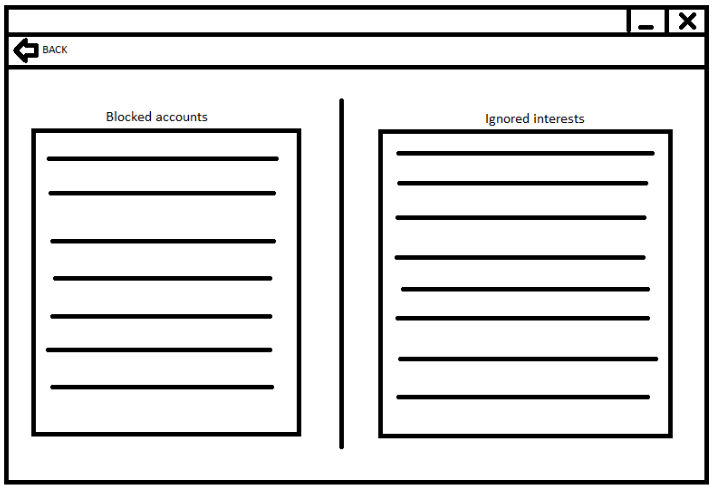
[2] CXL.com website – 10 Useful Findings About How People View Websites
[3] Healthline.com website – Is Dark Mode Better for Your Eyes?

Following stakeholder feedback, I changed the design to be more user friendly. I removed the text, as this hinders non-English speaking users and may be too small to read for some users. The numbers allow the user to enter the authentication code instead, drawing from Apple's passcode design and are large enough to be viewed easily.



This is my design for my recommendation GUI. As you can see, the accounts are displayed vertically, can be expanded to show more information, and has the required buttons. The settings button is displayed on top-left to stand-out.





The settings GUI is designed like this as it displays the key information in the centre of the screen. A back button is present top left to stand out and takes the user back to the recommendation GUI.

Key Variables, Data Structures and Classes

Name	Type	Validation	Explain (What it stores)	Justify (Why it's needed)
Global Variables				
gui	Object		Holds the object that represents the window displayed to the user.	To draw to the window, a reference of the GUI object must be stored to use the functions needed to render an image.
api	Object		The Twitter API instance.	The API is a key aspect of my program, as it allows me to read data from Twitter's servers and is used throughout all my program. As a result, a global reference is kept so that each time I need the API, I do not need to authenticate again.
Login Module				
authFile [Global]	File object	If the file is null, initialise the object, otherwise errors will be thrown when trying to load the data from the file.	Stores a representation of a file. The auth file will store the token and secret for the API permanently.	To read/write from the file, a reference should be kept. It is needed so authentication is only required once.
firstTime	Boolean		Whether the user has authenticated and used the application before.	The program needs to know whether it should make the user authenticate.
token	String	If string is empty, firstTime variable is set to false, therefore this does not need validation.	The authentication token used by the API.	If the user has authenticated before, this string must be provided to the API to access the features necessary for my program to function.
secret	String		The authentication secret used by the API.	
code	String	The code entered must be 6 digits long and can only contain alphanumeric characters, because this is the only pattern that Twitter	The six-digit code that the user enters to authenticate with Twitter.	If the user has not authenticated before, this code must be provided to the API to confirm that the user has requested access for its use.

		provides and so will only accept codes like this.		
Interests' Module				
tweets	Array of objects		A list of all tweets to process (which will either be from the user's feed or from an account's posts).	Rather than requesting/fetching a tweet from the API, waiting for a response, and processing it before fetching another, this array can be used as a buffer to fetch tweets and adds the tweet to the array in the background while the tweet at the start of the array is being processed. Like pipelining in the CPU.
tweet	Object		The current tweet from the array that is being processed.	As one tweet from the array is processed at a time, a reference of this tweet is held temporarily to manipulate (specifically applying the keyword extraction algorithm to).
i	Integer	Should not be less than zero or greater than length of tweets array. Otherwise, an index out of bounds exception will occur and the program will crash.	The iterative counter of the tweets array.	Needed to iterate over the tweets array, processing a singular tweet at a time.
listOfInterests	Array of strings		As tweets are processed, the extracted keywords/interests are stored in this array.	The recommendation algorithm requires a list of interests, which must be returned all at once.
Recommendation Module				
listOfInterests	Array of strings		The interests that the user has.	Each account's interests must be compared against the users to determine a match. It is more efficient to process this data once and store it as a variable for multiple uses rather

				than redefining a list of interests for every account.
listOfAccounts	Array of objects		The Twitter account representations to iterate over and compare.	This list can be used as a buffer, like above, to fetch accounts while another process is comparing the interests of the user to the current account.
account	Object	If the account is null, return. Otherwise, a null pointer exception will be thrown, and the program will crash.	Current account to process and compare interests with users.	As one account is processed at a time, a reference is temporarily held.
i	Integer	Should not be less than zero or greater than listOfAccounts length. Otherwise, an index out of bounds exception will be thrown and the program will crash.	The iterative counter of the listOfAccounts array.	Needed to iterate over the accounts array, processing a singular account at a time.
interestsOfAccount	Array of strings		The interests that the account has.	Used to compare the similarity of the users interests to the account.
threshold [Global]	Float/Real	Should not be less than zero or greater than one. If it is less than zero, the algorithm won't work as intended and recommendations will be inaccurate.	The threshold at which to start recommending the account. A constant variable.	If the similarity weighting is above this threshold, the account will be recommended to the user.
similarity	Float/Real		The value/weighting of how similar the user's interests are to the interests of the account. A higher value represents more similar interests.	Needed to compare how similar the interests are and determine whether this account should be recommended to the user.

recommendationList	Array of objects		Stores the accounts that should be recommended to the user.	As the algorithm processes more than one account at a time, there can be multiple matches and thus an array is needed to store multiple recommendations. This is returned to the calling function (the recommendation GUI to be displayed to the screen).
Settings Module				
settingsFile [Global]	File object	If file does not exist, create it. If the file is null, then the object must be instantiated otherwise a null pointer error will occur, and the program will crash, and the settings will not be able to be loaded.	The file representation of the settings of the user.	To keep settings throughout restarts, the settings need to be non-volatile. Therefore, the settings should be written to a file to save these.
listOfBlocked	Array of objects		A list of accounts that the user has blocked.	Needed so that the user can manage who they have blocked, adding, or removing from.
listOfIgnored	Array of strings		A list of interests/keywords that the user has ignored.	These keywords will have lower weighting when comparing the similarity of interests, and therefore, will be less likely to be recommended.
Keyword Extraction Algorithm				
stopwords	Array of strings	Must validate that stopwords array is not null. If it is null, assign it to an empty string array. If the array were to be null, this could cause an	This array stores a list of strings that the algorithm should ignore when iterating over the terms.	These stopwords are insignificant words that we do not want to spend processing time on (such as 'and', 'the', 'then', etc) as these words do not give an indication of interests. Therefore this array is needed as it will save processing time, making the algorithm more

		error in the algorithm when it checks if the array contains a word to compare if it is a stop word or not.		efficient and providing a better user experience.
termFrequencyMap	A key-value pair map		Stores the term (keyword/interest) and the frequency at which it appears in a set of texts.	Each word/term appears a certain number of times within a text. This map records the frequency at which a term appears and will be used to determine interests (terms with higher frequency).
wordCount	Integer		Counts the number of words processed so far from all given texts.	Used to calculate the relative term frequency (R) of a word. $R = F/C$.
threshold [Global]	Real/Float	Should not be less than zero or greater than one. If it is less than zero, the algorithm won't work or will provide inaccurate results.	The threshold at which a relative term frequency (R) becomes important.	Terms with a R value above this threshold will be added to the list of key terms.
listOfKeywords	Array of strings		The list of key terms that have an R value above the threshold.	The keywords must be returned to the calling code and so are stored in an array to be returned all at once.

Data structures

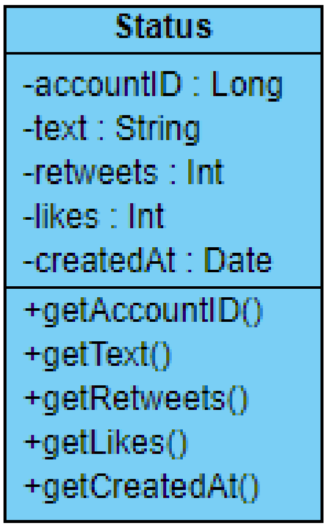
The most common data structure I used was an array. This is because often I have many variables of the same data type, which can therefore be collected into an array. An array allows iteration over the values, which is useful for my program to process one value at a time.

A key-value map/hash table, or more specifically in my chosen programming language of Java – a HashMap, stores keys and values in a pair. The HashMap is very efficient as the key is hashed to produce the index in the underlying data structure and therefore has quick store/retrieval speeds of time complexity O(1).

For the depth-first search (DFS), a stack is the most common implementation and therefore will be used for my searching algorithm. A stack if a first-in-first-out (FIFO) data structure, and therefore the first value entered the stack from the graph will also be the first to be traversed, producing a depth-first search.

Classes

A single tweet is encapsulated by Twitter4j's API class, called Status. This links to the documentation (or 'JavaDocs') of the class and explains each method.



Here is a summary of the important attributes and methods of the class, which will be useful for my program.

The accountID uniquely identifies the author of the tweet.

In addition to tweets, the API also encapsulates an accounts profile. This provides access to basic account information, such as display name, biography, profile picture, etc and will be used throughout my program.

Iterative Testing

Login Testing

This data will test whether the login is working properly. If a token is present, the application will authenticate using that data, if it is not present, the user should be told to authenticate. The user provided authentication code should then be tested, with different data types and inputs. It should only be successful if the code matches with Twitter's and successfully authenticates.

This test data is to ensure that the user has entered the code in the correct format, doing user validation will allow me to inform the user if they have entered an invalid code before using the Twitter API to test the code. This is client-side validation over server side to save processing power for Twitter and to decrease the time taken for a response as the latency to Twitter servers will not occur due to the client-side user validation.

Testing	Type	Data	Expected outcome
Token is empty	I	Token = “”	User told to authenticate
Token is valid	N	Token = “Gs5Xa”	Success
Auth code is valid	N	Code = “123456”	Success
Auth code is text	E	Code = “Apples”	Error thrown
Auth code is wrong	B	Code = “000000”	Error thrown
Auth code is invalid	E	Code = “.../*&”	Error thrown

I am using this test data as it covers all possibilities that the user could enter with varying success.

Keyword Extraction Algorithm Testing

The algorithm should only be used to process texts, such as a tweet. It should not do anything if anything other than a string/tweet is provided. This test data is to test the effectiveness of a keyword extraction algorithm, as there are multiple algorithms that perform that same task, with varying time and space complexities, test data must therefore be created to compare these algorithms will the same data.

Testing	Type	Data	Expected outcome
Text is empty	I	Text = “”	Nothing
Text is numbers	I	Text = 3	Error thrown
Text is valid	N	Text = tweet	Term frequency determined
Text is null	E	Text = null	Error thrown

I am using this test data as it covers all possibilities for strings that the algorithm will encounter and therefore must be coded to handle correctly.

Interest Testing

This test data will help determine whether the value passed in is valid or not. This data is to test the user input when searching for a specific interest. This will be user inputs and so must be validated to ensure the user has entered a correct input before performing an expensive/resource intensive algorithm. This will save performance for the user's computer. This user validation is client-side.

Testing	Type	Data	Expected outcome
String is passed	N	Data = “Football”	Success
Integer is passed	E	Data = 3	Error thrown

Float is passed	E	Data = 3.1	Error thrown
Non-alphabetic character is input	E	Data = /.,” %&	Error thrown

This test data is being used as it covers all possible types on input from the user. If the user enters an input that a result cannot be found for, a message will be displayed as this input test case is valid but will not have a useful result

Post-Development Testing

Test Scenario 1: New user (Emma)

- Open the program
- Authenticate with Twitter
- Enter correct code
- Interact with recommendations
- Open in browser
- Close the program

Test Scenario 2: Average user (Chirag, Sam)

- Open the program
- Authenticate
- Enter code
- Interact with recommendations
- Block account
- View more information about an account
- Search for a specific interest
- Open in browser
- Close the program

Test Scenario 3: Advanced user (Fin)

- Open the program
- Authenticate
- Search for specific interest regarding trending topics
- Open in browser
- View metrics
- Expand information
- Search for different interest
- Open settings GUI
- Unblock user
- Ignore certain interest
- Return to recommendations
- Search for ignored interest
- Close the program

Justification for post-development testing/scenarios

These test scenarios test all aspects of the program for different levels of user/Twitter experience. Therefore, they are appropriate for my stakeholders as they will give valuable feedback into the design of my program if the stakeholder finds these tasks simple to carry out or not.

Test scenario one gives insight of a basic level of user and has basic interactions. This is measurable as the user should be able to carry out this task easily and quickly without any confusion into how to complete the next step.

Test scenario two gives insight into an average level of user. This scenario has more steps to test the user’s ability to use the program. Again, all steps should be easy to complete, and the user should not struggle to complete this. If the user can complete this, the design is successful as it is simple to use.

Test scenario three give insight into a highly advanced level of user. This scenario has more complex tasks that most basic to advanced users will not interact with. Some of these tasks to carry out may not be completed in the finished product as they are possible extras, not part of the success criteria. An advanced level of user should be able to complete and use these components of the program to aid their Twitter use. Again, the advanced user should still be able to easily carry out these tasks. This will be measurable by the time taken to complete this complex task.

This table should be filled in later, post development.

Stakeholder	**What worked?**	**What errors occurred?**	**How did these errors impact your experience?**	**What areas need improvement?**	**How successful was the system?**
Chirag					
Sam					
Emma					
Fin					

Development

The first module I will be developing is the login module. This is because to develop the other modules, such as the keyword extraction algorithm module, I require Twitter's API for things such as fetching tweets. However, I am unable to use the API's features without authentication, and therefore I must develop this module first in order to authenticate with the API.

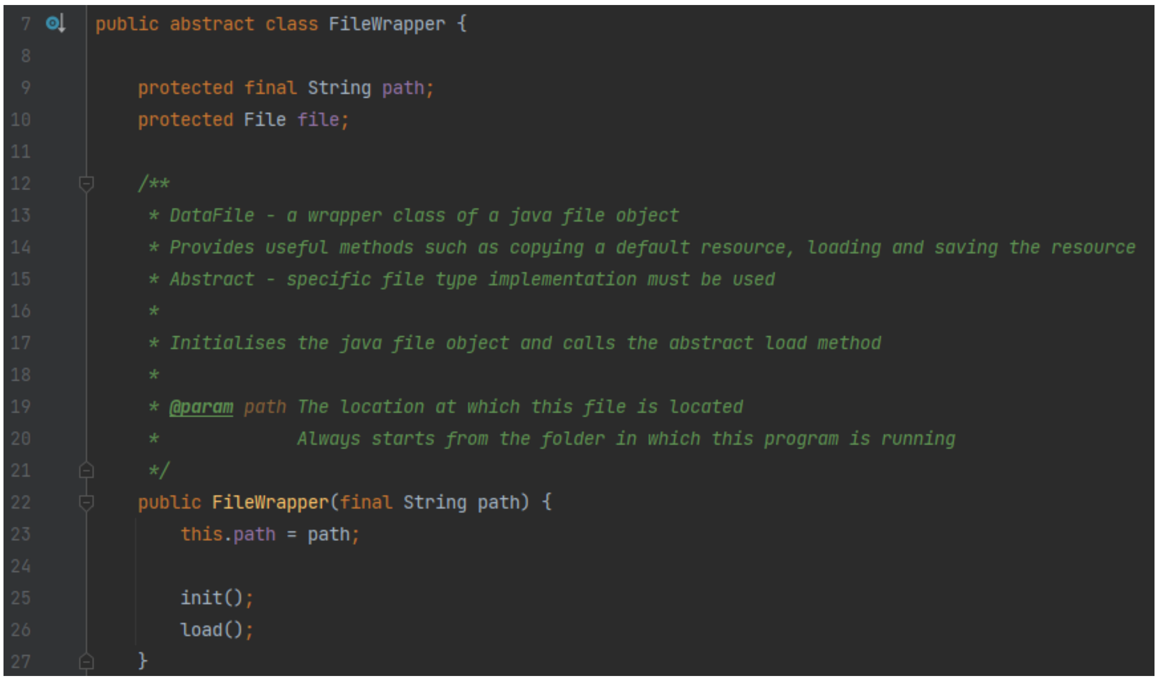
Firstly, I will develop the code that enables me to authenticate with the API, only afterwards once I have fully tested this will I implement it into a GUI. This allows me to focus on the login process more and less on the look, furthermore I will still be able to test sufficiently using the console.

Login module iterations

Iteration one

Firstly, as specified in my requirements, the login page should only show if the user has not previously authenticated. To check this, a file will be kept on the user's hard drive containing the authentication token and secret that need to be provided to the Twitter API.

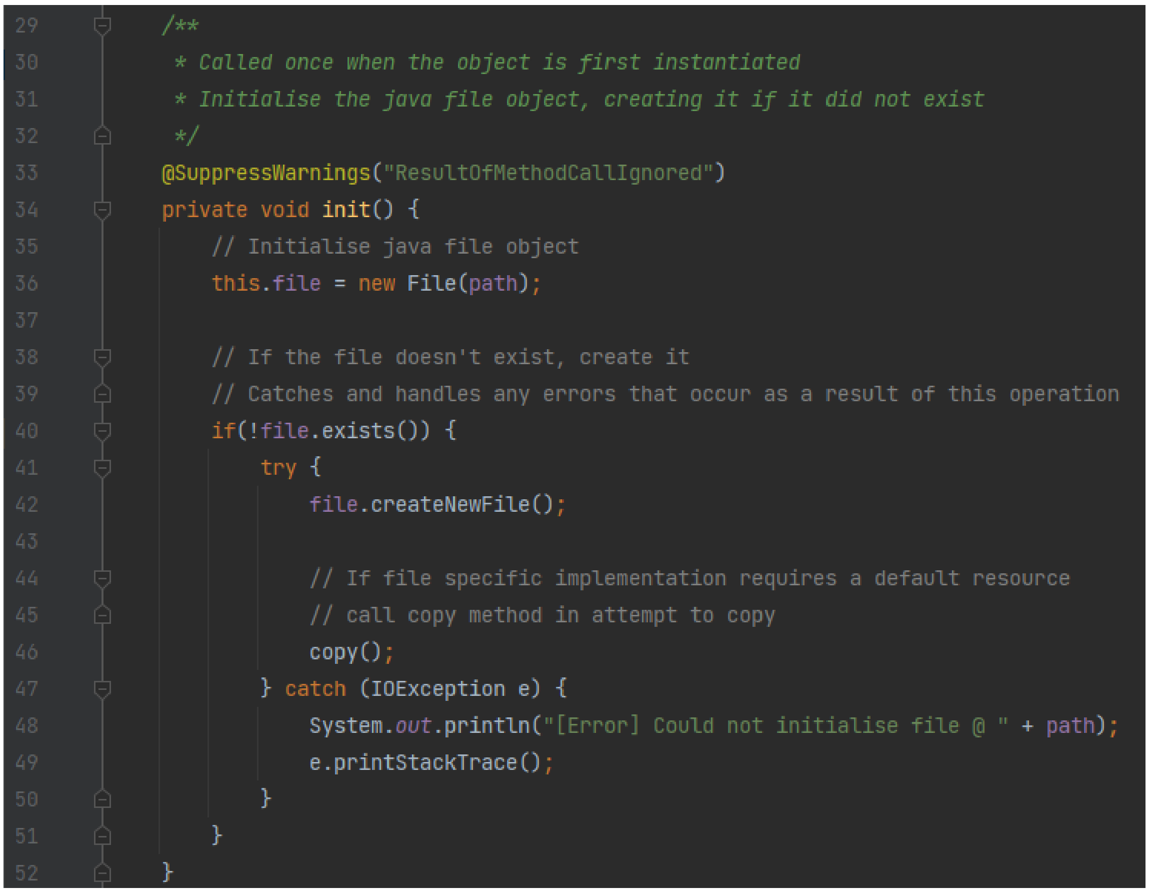
To implement this, I need a wrapper class – a wrapper class encapsulates the functionality of primitive data types or another class – to create a functional file interface that I can use throughout my program.



```java
public abstract class FileWrapper {

    protected final String path;
    protected File file;

    /**
     * DataFile - a wrapper class of a java file object
     * Provides useful methods such as copying a default resource, loading and saving the resource
     * Abstract - specific file type implementation must be used
     *
     * Initialises the java file object and calls the abstract load method
     *
     * @param path The location at which this file is located
     *             Always starts from the folder in which this program is running
     */
    public FileWrapper(final String path) {
        this.path = path;

        init();
        load();
    }
```

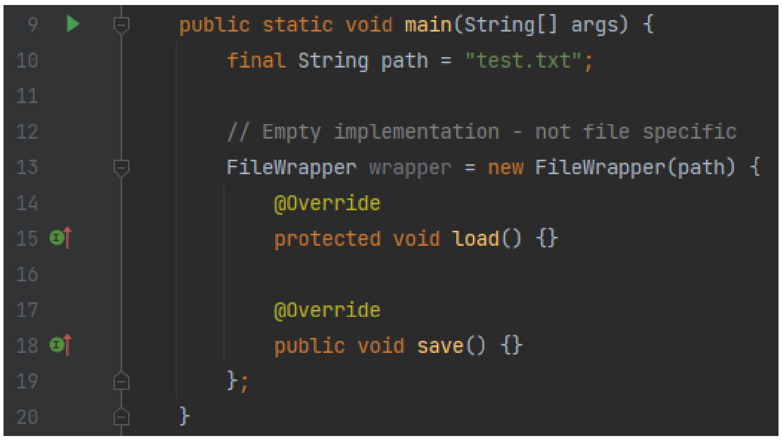
As this is a class, it can be instantiated multiple times and therefore my program can use multiple file objects, which is beneficial as I only have to code this once, but it can be used multiple times in my program.

The constructor takes in a path at which the file is located, and then calls the init method followed by the load method. In this init method, the file object is instantiated, and if the physical file does not exist on the user's hard drive, it is created.



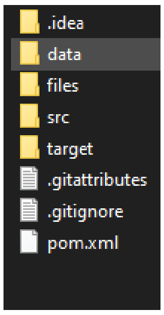
```java
    /**
     * Called once when the object is first instantiated
     * Initialise the java file object, creating it if it did not exist
     */
    @SuppressWarnings("ResultOfMethodCallIgnored")
    private void init() {
        // Initialise java file object
        this.file = new File(path);

        // If the file doesn't exist, create it
        // Catches and handles any errors that occur as a result of this operation
        if(!file.exists()) {
            try {
                file.createNewFile();

                // If file specific implementation requires a default resource
                // call copy method in attempt to copy
                copy();
            } catch (IOException e) {
                System.out.println("[Error] Could not initialise file @ " + path);
                e.printStackTrace();
            }
        }
    }
```

This code is a non-file type specific implementation of my file wrapper class, in order to test its functionality. Once the new method is called, the constructor goes on to call the init method, which should create the file as it does not exist.

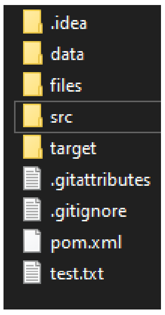


```java
    public static void main(String[] args) {
        final String path = "test.txt";

        // Empty implementation - not file specific
        FileWrapper wrapper = new FileWrapper(path) {
            @Override
            protected void load() {}

            @Override
            public void save() {}
        };
    }
```

Here are before vs. after screenshots of my project directory after creating a file:



```
.idea
data
files
src
target
.gitattributes
.gitignore
pom.xml
```



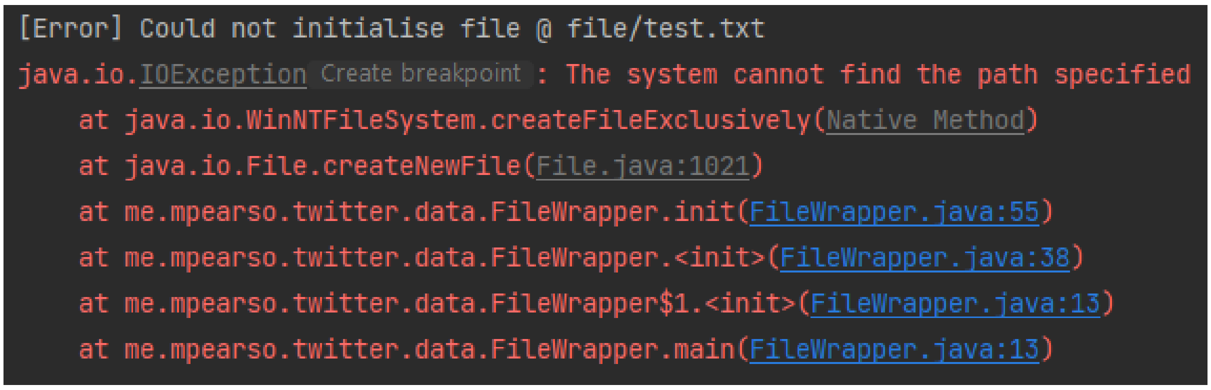
```
.idea
data
files
src
target
.gitattributes
.gitignore
pom.xml
test.txt
```

As can be seen at the bottom, the file "test.txt" has been created.

I continued with more test cases to review my code and ensure it was working correctly.

Test case	Expected result	Actual result
Creating file that does not exist, without directory	File is created successfully and named after the specified path	File is created as expected
Creating file that does not exist, with directory	Directory is created, file is created within and named after the specified path	An error was thrown, stating that the path at which the file was located could not be found
Retest after fixing the code to create the parent directories		Parent directory and file are created as expected
Creating a file that already exists, without directory	No change	No change
Creating a file that already exists, with directory	No change	No change
Creating file that does not exist, with multiple directories	All parent directories are created, file is created within and named correctly	File and parent directories are created as expected

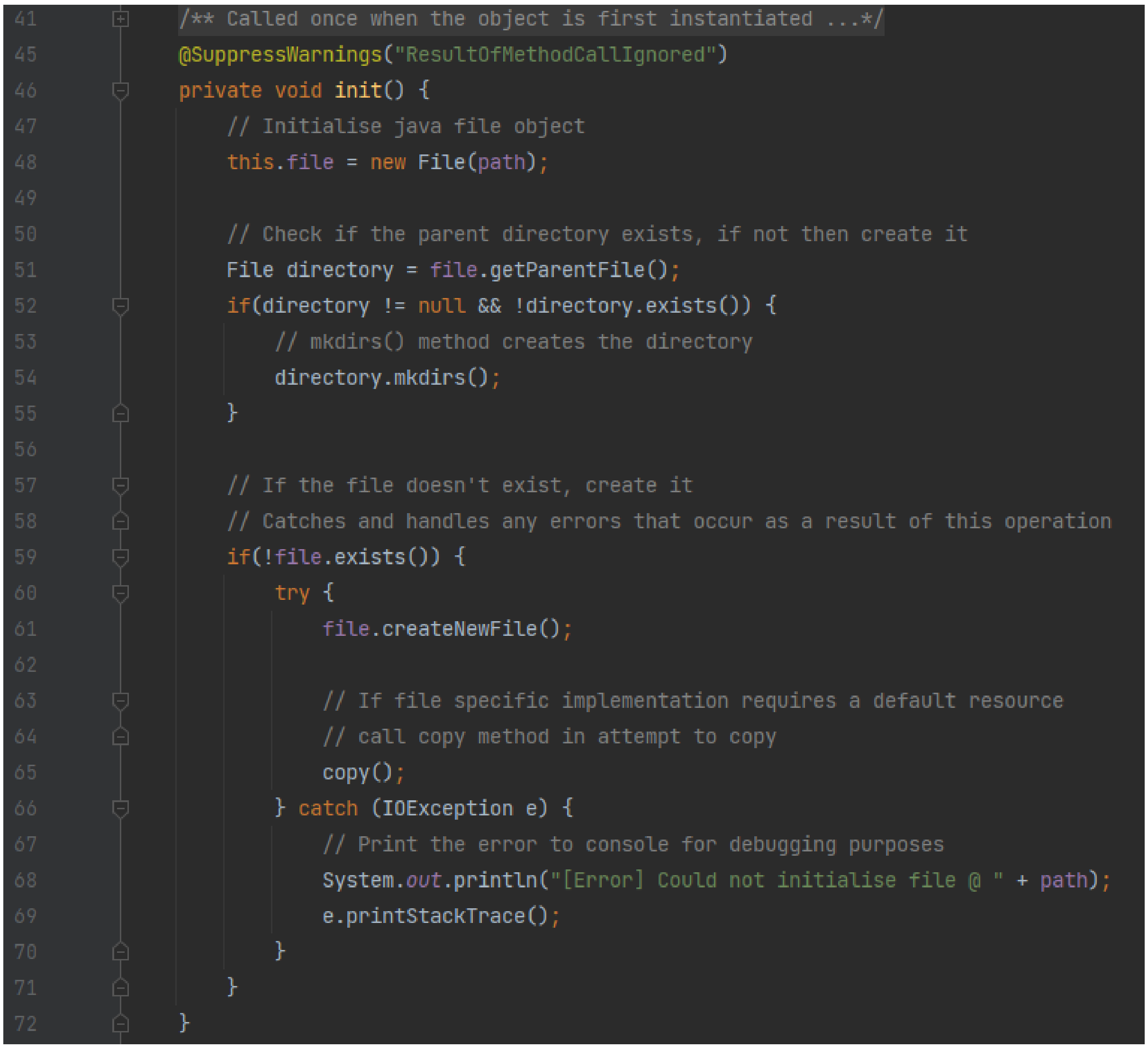
The image below shows the error thrown when trying to create a file at path "file/test.txt" while the directory "file" does not exist:



```
[Error] Could not initialise file @ file/test.txt
java.io.IOException Create breakpoint : The system cannot find the path specified
    at java.io.WinNTFileSystem.createFileExclusively(Native Method)
    at java.io.File.createNewFile(File.java:1021)
    at me.mpearso.twitter.data.FileWrapper.init(FileWrapper.java:55)
    at me.mpearso.twitter.data.FileWrapper.<init>(FileWrapper.java:38)
    at me.mpearso.twitter.data.FileWrapper$1.<init>(FileWrapper.java:13)
    at me.mpearso.twitter.data.FileWrapper.main(FileWrapper.java:13)
```

As the stack trace shows, the error was when I tried to create the new file, as the directory in which I had specified did not exist. After researching online, to overcome this I must first create the parent directories. To do this, I simply get the parent files/directories using a method in the java file object 'getParentFiles()' and call the method 'mkdirs()' to create any missing directories. My justification of this solution to this error is that it is the simplest way that is recommended to avoid this issue.

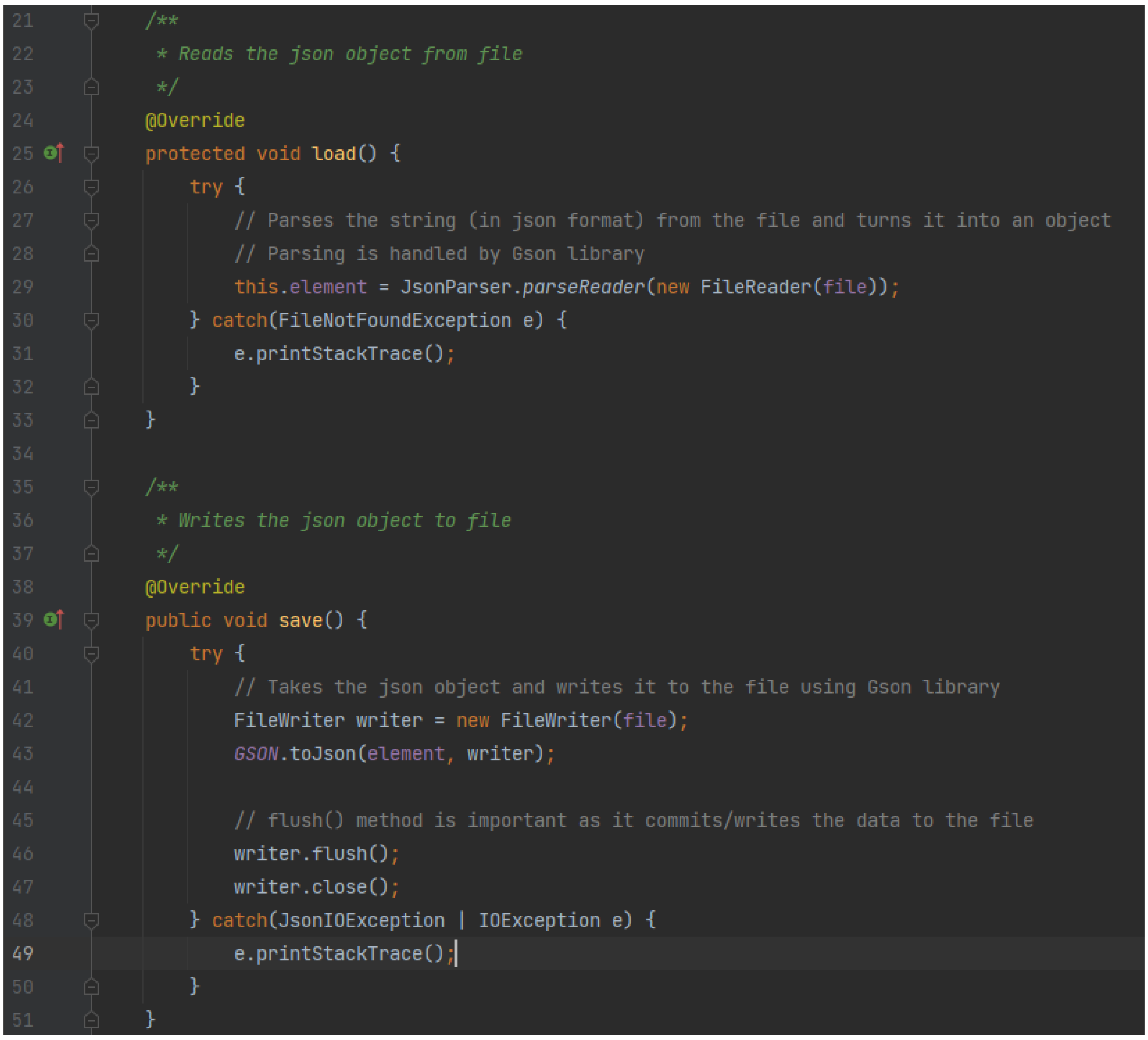
The fixed init method is shown on the next page, as can be seen from line 50-55, the method now validates whether the parent directories exist in order to prevent this error.



```java
/** Called once when the object is first instantiated ...*/
@SuppressWarnings("ResultOfMethodCallIgnored")
private void init() {
    // Initialise java file object
    this.file = new File(path);

    // Check if the parent directory exists, if not then create it
    File directory = file.getParentFile();
    if(directory != null && !directory.exists()) {
        // mkdirs() method creates the directory
        directory.mkdirs();
    }

    // If the file doesn't exist, create it
    // Catches and handles any errors that occur as a result of this operation
    if(!file.exists()) {
        try {
            file.createNewFile();

            // If file specific implementation requires a default resource
            // call copy method in attempt to copy
            copy();
        } catch (IOException e) {
            // Print the error to console for debugging purposes
            System.out.println("[Error] Could not initialise file @ " + path);
            e.printStackTrace();
        }
    }
}
```

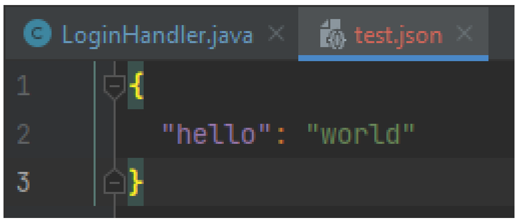
To meet requirement 1.5 from my analysis' break down, I will code the file specific implementation that I require to write the data to a file. For this, the authenticated data will be stored as a .json file type, as this format is easy to read from and write to. In addition, it is formatted using attribute-value pairs, which is suited to the nature of my data. To parse the json, I will be using an open-sourced library called Gson by Google.

Shown below is the implementation for loading and saving a JSON file.



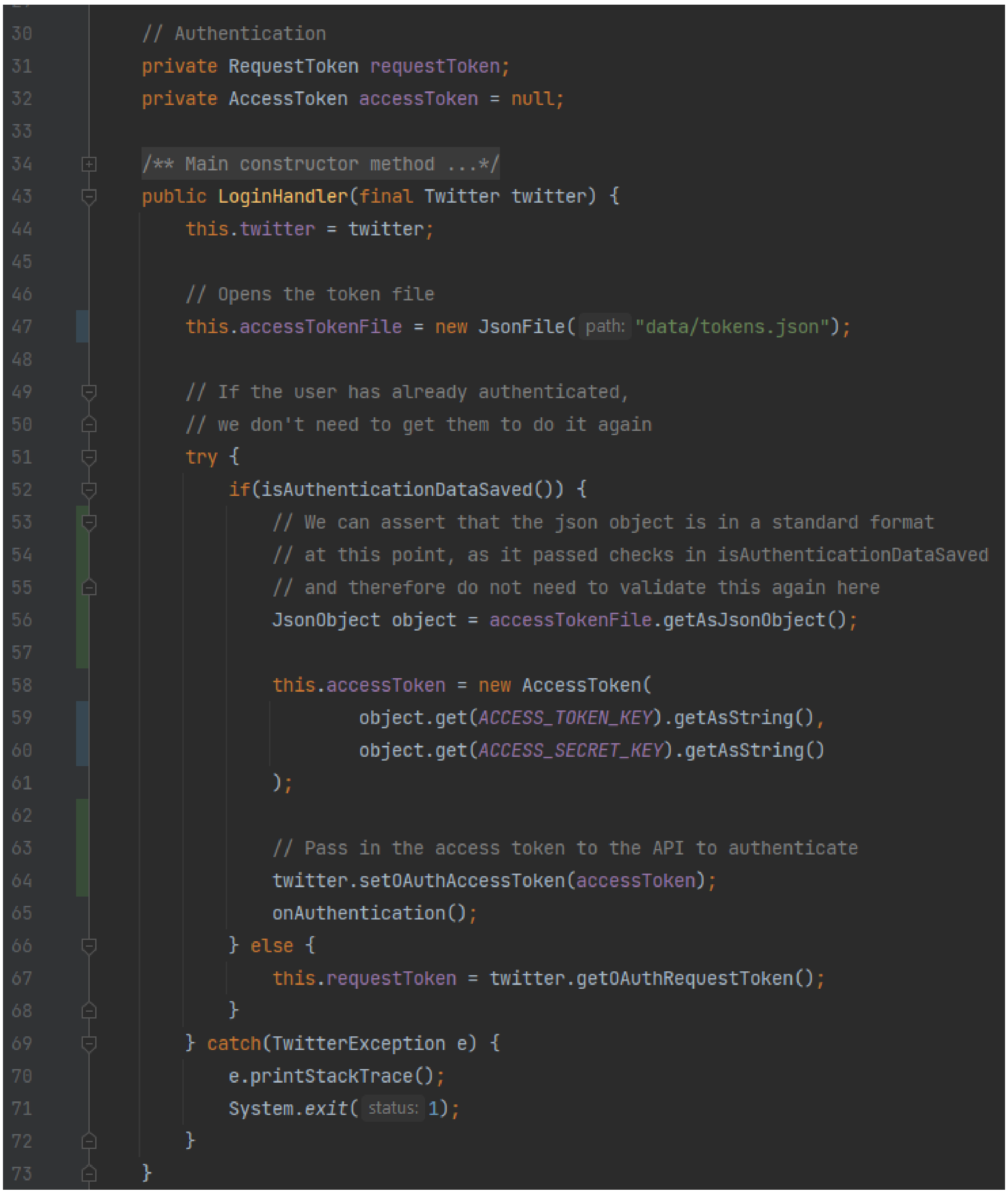
```java
    /**
     * Reads the json object from file
     */
    @Override
    protected void load() {
        try {
            // Parses the string (in json format) from the file and turns it into an object
            // Parsing is handled by Gson library
            this.element = JsonParser.parseReader(new FileReader(file));
        } catch(FileNotFoundException e) {
            e.printStackTrace();
        }
    }

    /**
     * Writes the json object to file
     */
    @Override
    public void save() {
        try {
            // Takes the json object and writes it to the file using Gson library
            FileWriter writer = new FileWriter(file);
            GSON.toJson(element, writer);

            // flush() method is important as it commits/writes the data to the file
            writer.flush();
            writer.close();
        } catch(JsonIOException | IOException e) {
            e.printStackTrace();
        }
    }
```

After testing the code, it successfully loads and saves to and from the file in a json format and the output can be seen here:



```json
{
  "hello": "world"
}
```

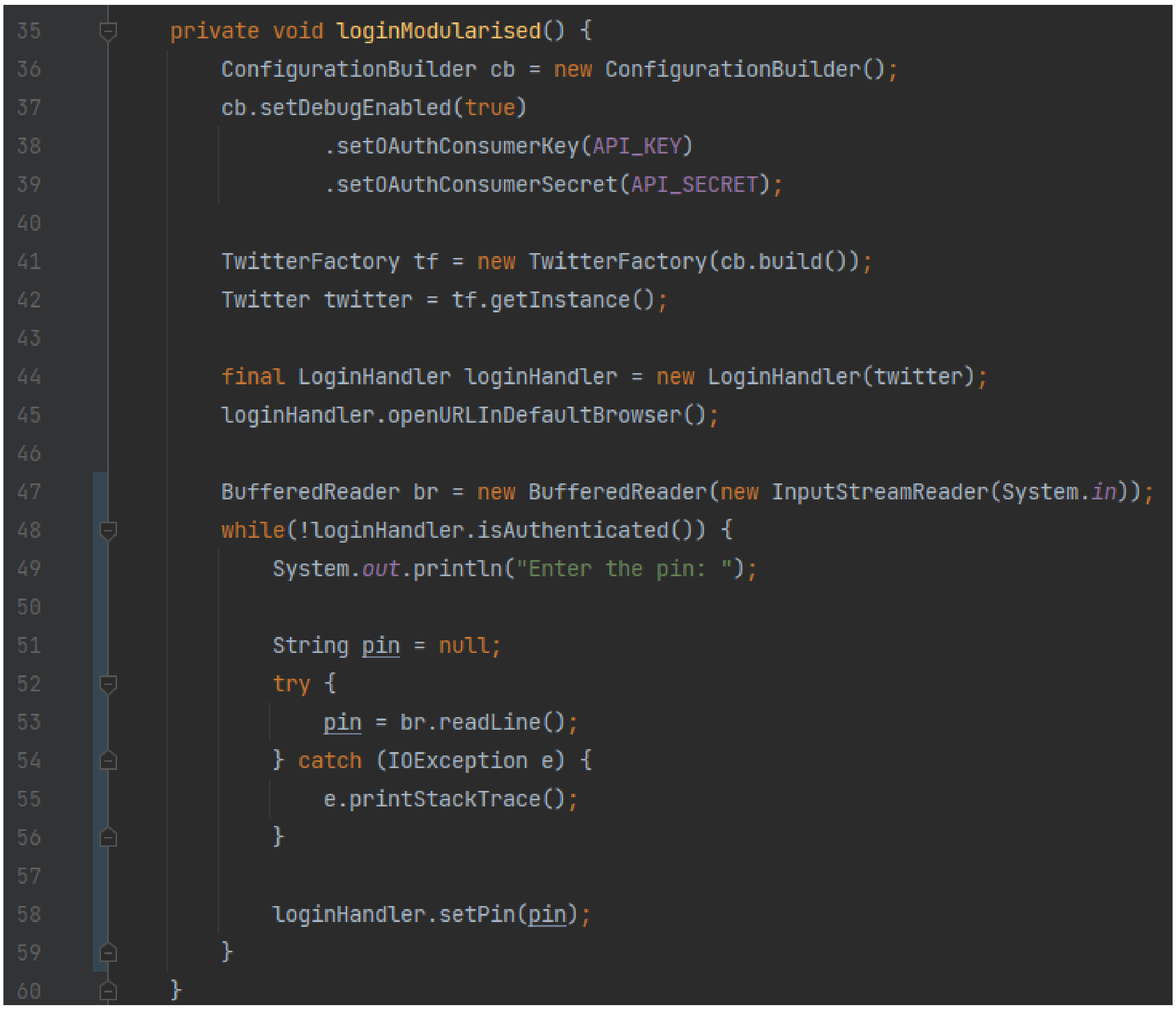
Finally, after coding the utility classes I will need for my login module, I can start the module itself. The class that handles the login process will be named LoginHandler and the constructor is as follows:



```java
    // Authentication
    private RequestToken requestToken;
    private AccessToken accessToken = null;

    /** Main constructor method ...*/
    public LoginHandler(final Twitter twitter) {
        this.twitter = twitter;

        // Opens the token file
        this.accessTokenFile = new JsonFile( path: "data/tokens.json");

        // If the user has already authenticated,
        // we don't need to get them to do it again
        try {
            if(isAuthenticationDataSaved()) {
                // We can assert that the json object is in a standard format
                // at this point, as it passed checks in isAuthenticationDataSaved
                // and therefore do not need to validate this again here
                JsonObject object = accessTokenFile.getAsJsonObject();

                this.accessToken = new AccessToken(
                        object.get(ACCESS_TOKEN_KEY).getAsString(),
                        object.get(ACCESS_SECRET_KEY).getAsString()
                );

                // Pass in the access token to the API to authenticate
                twitter.setOAuthAccessToken(accessToken);
                onAuthentication();
            } else {
                this.requestToken = twitter.getOAuthRequestToken();
            }
        } catch(TwitterException e) {
            e.printStackTrace();
            System.exit( status: 1);
        }
    }
```

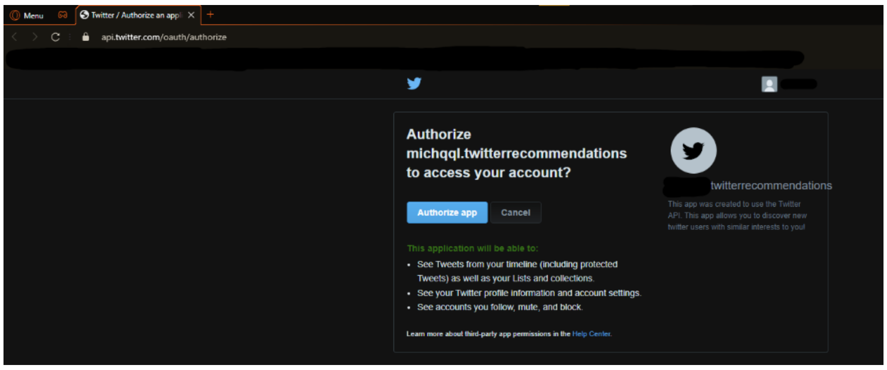
In this constructor method, the file that stores the authentication token and secret is opened, named accessTokenFile. Then, the program checks whether the file already contains the data that it needs to authenticate. This has been moved to a separate method, isAuthenticationDataSaved, as this code is reusable and will need to be checked by other areas of the program. If the data has been saved previously, the token to access Twitter's API is created and passed to the Twitter API instance to authenticate.

To test this login handler, I created this temporary test code to try and authenticate the user:



```java
35      private void loginModularised() {
36          ConfigurationBuilder cb = new ConfigurationBuilder();
37          cb.setDebugEnabled(true)
38                  .setOAuthConsumerKey(API_KEY)
39                  .setOAuthConsumerSecret(API_SECRET);
40
41          TwitterFactory tf = new TwitterFactory(cb.build());
42          Twitter twitter = tf.getInstance();
43
44          final LoginHandler loginHandler = new LoginHandler(twitter);
45          loginHandler.openURLInDefaultBrowser();
46
47          BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
48          while(!loginHandler.isAuthenticated()) {
49              System.out.println("Enter the pin: ");
50
51              String pin = null;
52              try {
53                  pin = br.readLine();
54              } catch (IOException e) {
55                  e.printStackTrace();
56              }
57
58              loginHandler.setPin(pin);
59          }
60      }
```

As can be seen on line 44, the login handler object is created and the method to open the authentication website in the user's default browser is called. This method works successfully.



[redacted]
[redacted]

Authorize michqql.twitterrecommendations to access your account?

[redacted]twitterrecommendations

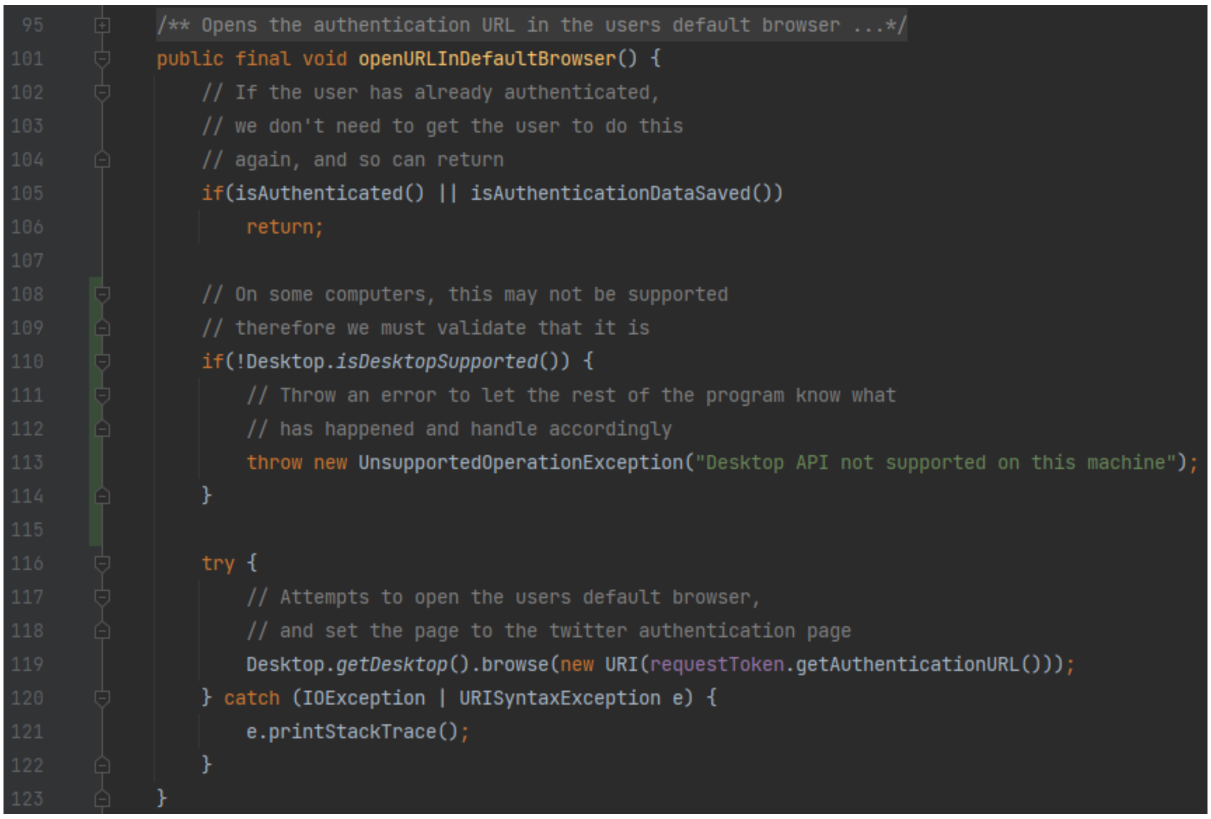
This app was created to use the Twitter API. This app allows you to discover new twitter users with similar interests to you!

This application will be able to:

- See Tweets from your timeline (including protected Tweets) as well as your Lists and collections.
- See your Twitter profile information and account settings.
- See accounts you follow, mute, and block.

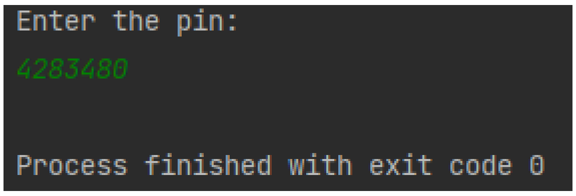
Learn more about third-party app permissions in the Help Center.

To accomplish this, I used Java's in-built API to open the browser associated with the system default. This links to requirement 1.2 in my analysis, as the default browser must be used to ensure the user is satisfied and feels safe using the browser.



```java
/** Opens the authentication URL in the users default browser ...*/
public final void openURLInDefaultBrowser() {
    // If the user has already authenticated,
    // we don't need to get the user to do this
    // again, and so can return
    if(isAuthenticated() || isAuthenticationDataSaved())
        return;

    // On some computers, this may not be supported
    // therefore we must validate that it is
    if(!Desktop.isDesktopSupported()) {
        // Throw an error to let the rest of the program know what
        // has happened and handle accordingly
        throw new UnsupportedOperationException("Desktop API not supported on this machine");
    }

    try {
        // Attempts to open the users default browser,
        // and set the page to the twitter authentication page
        Desktop.getDesktop().browse(new URI(requestToken.getAuthenticationURL()));
    } catch (IOException | URISyntaxException e) {
        e.printStackTrace();
    }
}
```

Finally, when the user enters an input, the authentication pin is set, and the program attempts to authenticate with the API. Currently, this is done through the console, as the GUI will be developed in a later iteration. As shown below, the program successfully exits when the correct pin has been provided.



```
Enter the pin:
4283480

Process finished with exit code 0
```

Validation of key element

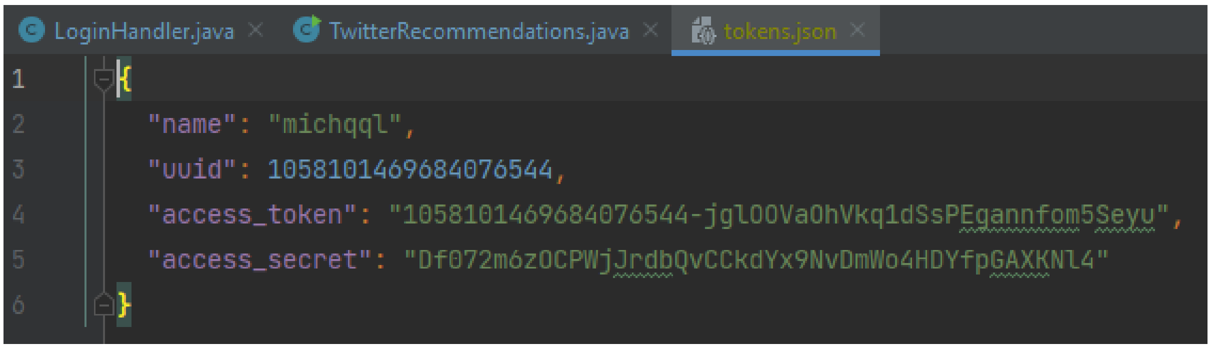
To validate and review this code, I will input a variety of test data to compare the expected vs actual results and make any changes accordingly.

Test case	Expected result	Actual result
The pin provided by Twitter's website is input (pin is randomly generated on the website)	Successful authentication and authentication data is saved to file	Result as expected

No pin is input – “”	Error will be thrown as the API could not authenticate	Error thrown as expected
Incorrect pin is input – “helloworld123456”	Error will be thrown as the API could not authenticate	Error thrown as expected

Currently, it is working as expected. Further along in the development process, these errors will be validated and handled by the GUI, as this will allow my program to inform the user of the error and how to properly input the correct pin.

After providing the correct pin, the access tokens and secret are stored to the “tokens.json” file, ready to be used when next launching the program, as shown below.



```
{
  "name": "michqql",
  "uuid": 1058101469684076544,
  "access_token": "1058101469684076544-jglOOVaOhVkq1dSsPEgannfom5Seyu",
  "access_secret": "Df072m6zOCPWjJrdbQvCCkdYx9NvDmWo4HDYfpGAXKNl4"
}
```

At this point, the first iteration of the login module has been completed, as my initial goal to authenticate with the Twitter API at the most basic level has been accomplished. I am now able to develop other parts of my program as I will have access to the required API methods and components.

Review

From this first iteration of the login module, I will mark requirements that have been partially met in yellow/amber or fully completed in green. Currently, while only one requirement has been met at this first iteration, two others are in progress. The rest rely on the GUI and so will be completed in a later iteration.

Requirement Number	**Requirement**	**Justification**	**Success Criteria & Evidence**	**Provided Evidence**
Login page/module				
1.1	Login UI is only shown to user if they are not logged in or it is the first-time using application	The user only needs to authenticate with Twitter once, and therefore this page only needs to be shown to the user once	Login UI is displayed first-time, but not again once they have authenticated. Screenshots of the menu taken to once loaded with and without login data.	

🟧 1.2	Button to open authentication in default browser	Using twitters website, the user must get an authentication code. Furthermore, using the default browser the user feels safe as this is their choice in browser	Clicking button opens users default browser to Twitter's authentication website. Screenshot of correct website	Upon running the program, the default browser is opened to the authentication website, as shown in screenshots above
1.3	Number pin field (buttons)	Allows user to enter the authentication code provided by Twitter	Screenshot of buttons that user can press + their effect once pressed	
🟧 1.4	Logs in once valid authentication code is provided	The code will be used to log in with Twitters API and the rest of the application can then be used. The user will be taken to the main screen	Entering correct code changes UI screen to main recommendations. Screenshot of before + after entering correct code	Upon entering correct pin, no error is thrown, and program exits. As shown in screenshots and test cases above
🟩 1.5	Authentication data is saved to file	This is required so that the user doesn't need to authenticate each time the load the application	After the user has authenticated for the first time, the data regarding their account is stored to a file on their computer. Screenshot of stored data	Data relating to the user, the token and secret are all saved to "tokens.json". As shown in screenshot above

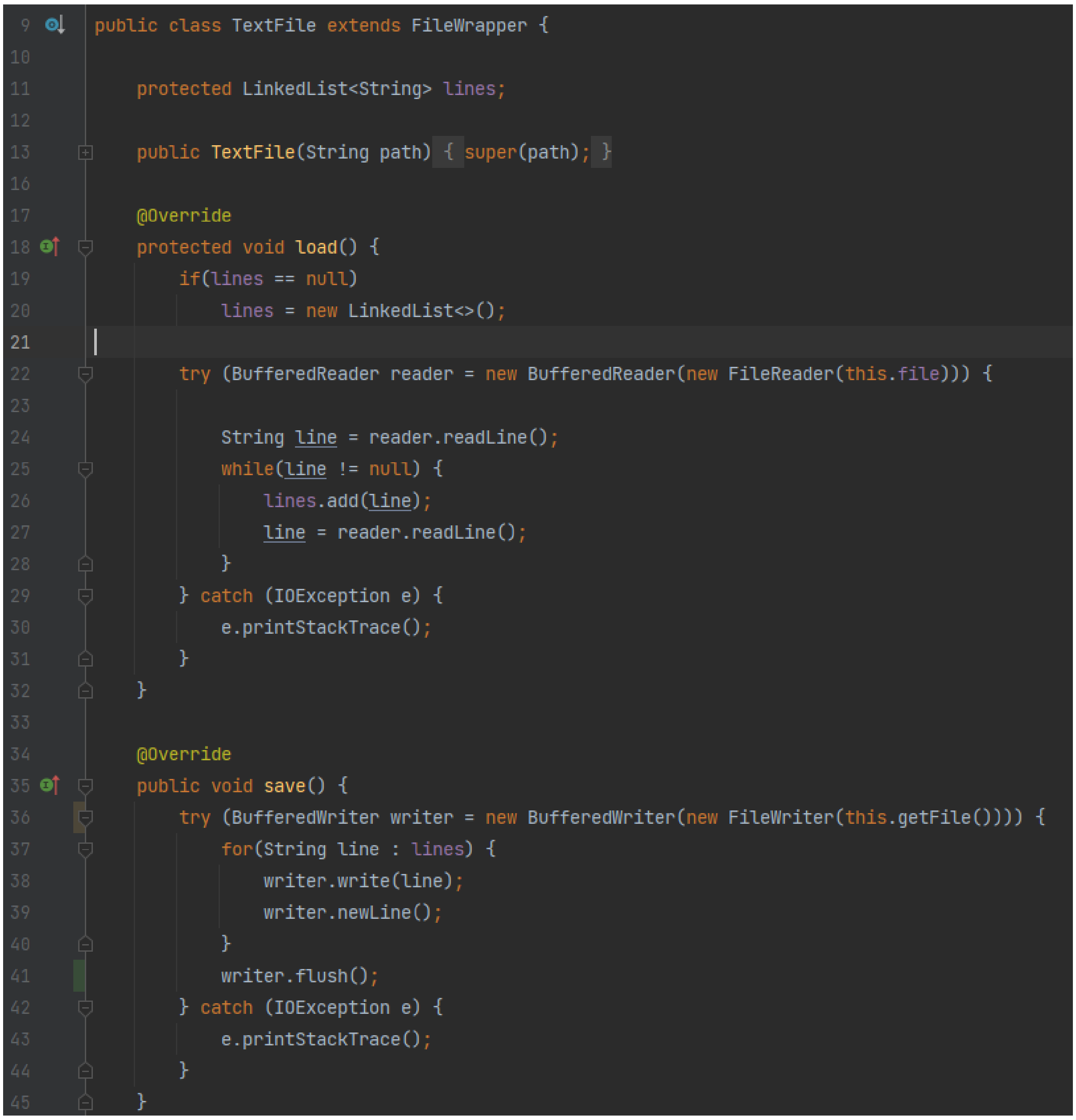
Recommendation module iteration

The second module to be developed will be the main part of my program. I will start by coding a few different variations of the keyword extraction algorithm and comparing the results and efficiency with each other to determine the best suited for my program.

Iteration two – simple algorithm

The standard keyword extraction algorithm takes a text, removes any ‘common’ stopwords and returns a term frequency map. I will code this algorithm first as it is the most simple and easy to code. In the case that I do run out of time on this project, as I have coded this simple base algorithm at least I will have a working program (however, it may not be the most efficient or accurate).

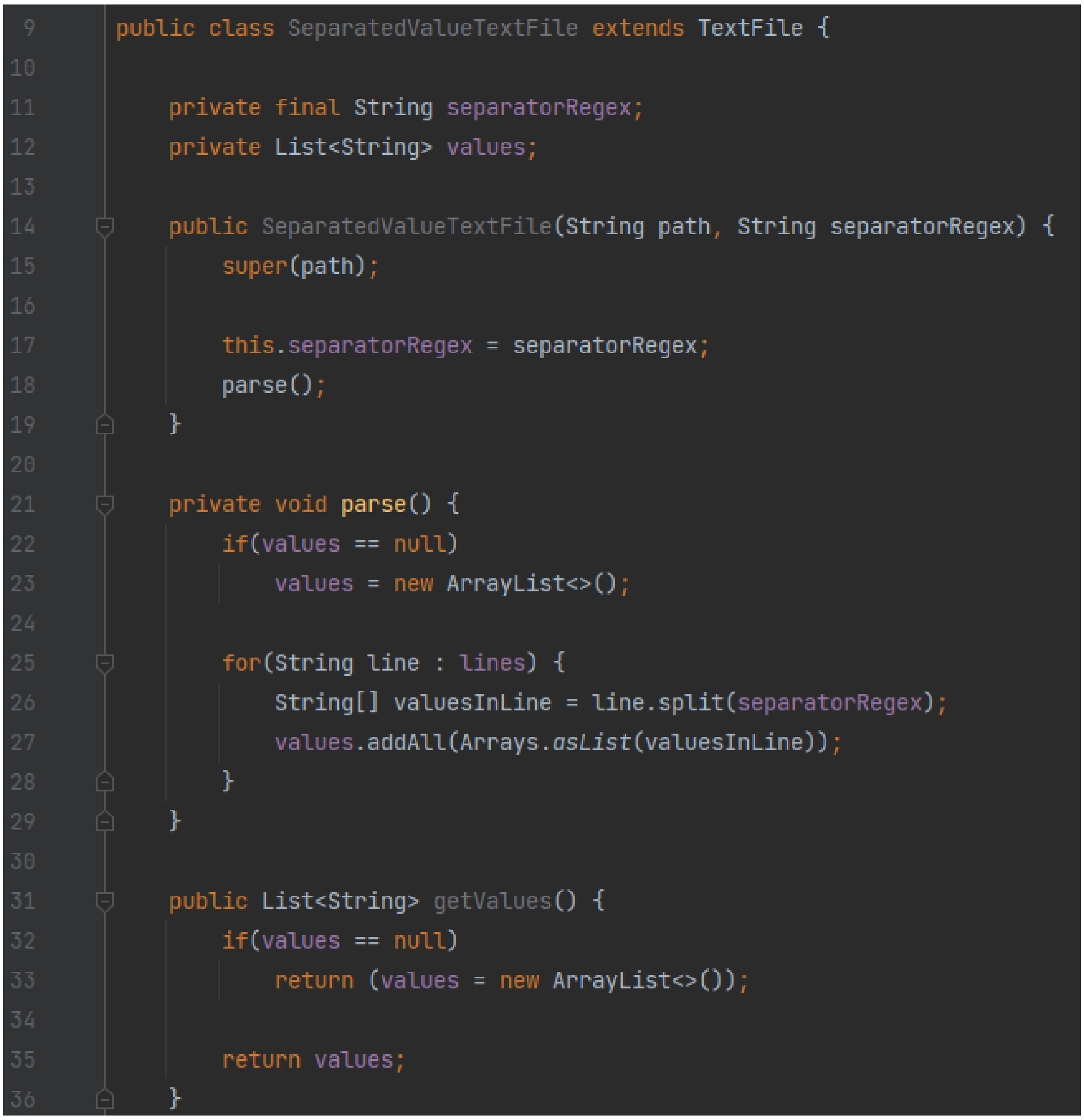
Firstly, I downloaded a list of stop words from online. Stop words are terms (words) that the algorithm should avoid. Terms such as ‘and’, ‘the’, ‘then’, etc. These provide no information about a user’s interests and so can be skipped over to avoid waiting processing time on these unnecessary terms. To load these stopwords in, I can use polymorphism and extend my ‘FileWrapper’ class to read and write with a ‘.txt’ file. Below shows this implementation:



```
public class TextFile extends FileWrapper {

    protected LinkedList<String> lines;

    public TextFile(String path) { super(path); }

    @Override
    protected void load() {
        if(lines == null)
            lines = new LinkedList<>();

        try (BufferedReader reader = new BufferedReader(new FileReader(this.file))) {

            String line = reader.readLine();
            while(line != null) {
                lines.add(line);
                line = reader.readLine();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void save() {
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(this.getFile()))) {
            for(String line : lines) {
                writer.write(line);
                writer.newLine();
            }
            writer.flush();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

Shown in the picture, this class reads in the text file line by line and adds it to a linked list data structure. To write to the text file, the class loops through the linked list and writes it to the text file, creating a new line after each. Finally, the writer is flushed. This pushes the data to the disk and ensures there are no problems further on in development with missing data.

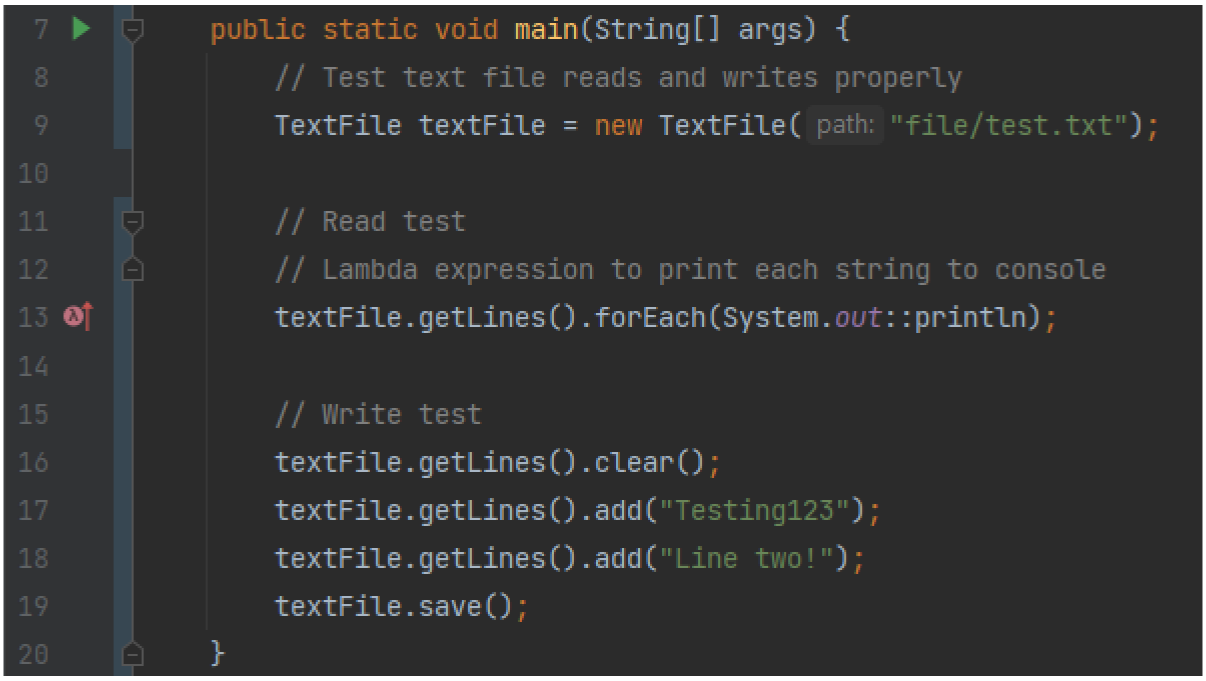
Using this class, I can then create another sub-class. This time, the class will read the data line by line and separate it via a separator character (such as a comma). This will create a simple implementation of comma separated value files.



```java
public class SeparatedValueTextFile extends TextFile {

    private final String separatorRegex;
    private List<String> values;

    public SeparatedValueTextFile(String path, String separatorRegex) {
        super(path);

        this.separatorRegex = separatorRegex;
        parse();
    }

    private void parse() {
        if(values == null)
            values = new ArrayList<>();

        for(String line : lines) {
            String[] valuesInLine = line.split(separatorRegex);
            values.addAll(Arrays.asList(valuesInLine));
        }
    }

    public List<String> getValues() {
        if(values == null)
            return (values = new ArrayList<>());

        return values;
    }
```

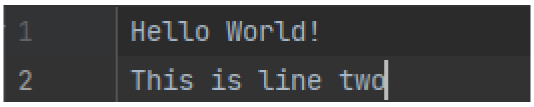
As can be seen in both classes, there is validation for the array being null before trying to add to or return it. This prevents any null pointer exceptions from occurring and potentially crashing my program.

To test these classes worked as intended, I wrote this code to read from and write to a file:



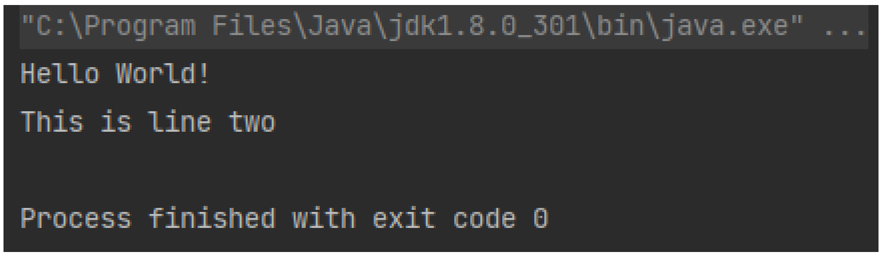
```java
    public static void main(String[] args) {
        // Test text file reads and writes properly
        TextFile textFile = new TextFile( path: "file/test.txt");

        // Read test
        // Lambda expression to print each string to console
        textFile.getLines().forEach(System.out::println);

        // Write test
        textFile.getLines().clear();
        textFile.getLines().add("Testing123");
        textFile.getLines().add("Line two!");
        textFile.save();
    }
```

This is the file before running the test function:



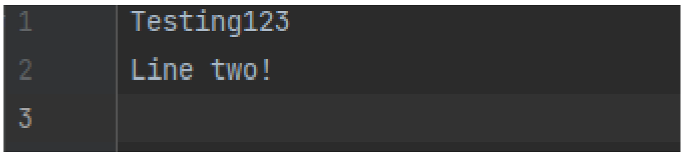
```
Hello World!
This is line two
```

Running the code, the lines are printed to the console as expected and the program exited with no issues (indicated by exit code 0)



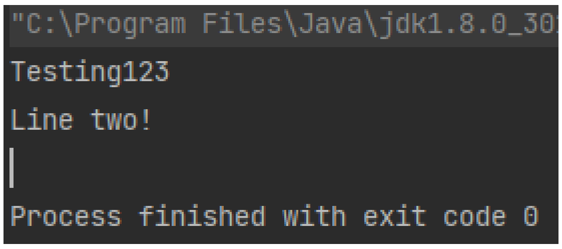
```
"C:\Program Files\Java\jdk1.8.0_301\bin\java.exe" ...
Hello World!
This is line two

Process finished with exit code 0
```

Furthermore, upon checking the text file again, it can be confirmed that the test function successfully wrote the new data to the file and to disk. The only possible issue that could occur with this is that a final new line is inserted at the very end of the document. This could cause an empty string to be loaded and this could cause issues in parts of my program.



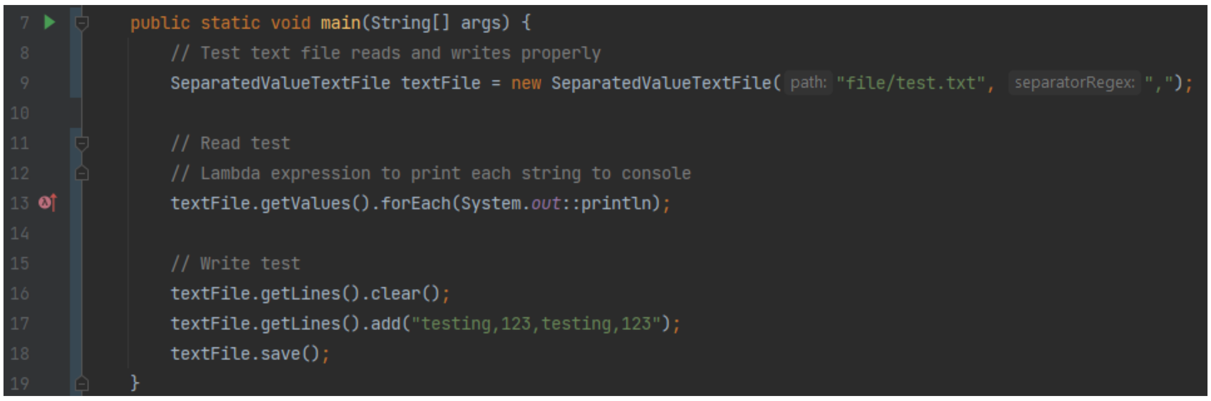
```
1  Testing123
2  Line two!
3
```

Upon running the program again however, it can be seen that this new/empty line was not read as there was only one space in the console (the same as the test run before).



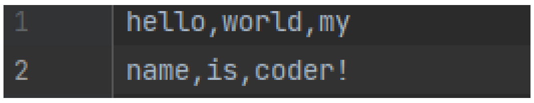
```
"C:\Program Files\Java\jdk1.8.0_30
Testing123
Line two!

Process finished with exit code 0
```

To test the separated values text file implementation, I wrote this test function:



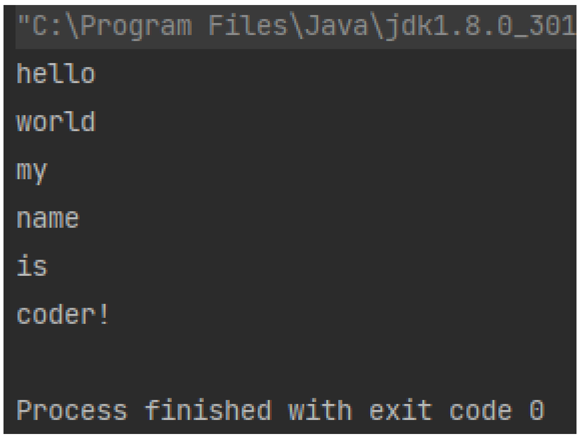
```java
public static void main(String[] args) {
    // Test text file reads and writes properly
    SeparatedValueTextFile textFile = new SeparatedValueTextFile( path: "file/test.txt",  separatorRegex: ",");

    // Read test
    // Lambda expression to print each string to console
    textFile.getValues().forEach(System.out::println);

    // Write test
    textFile.getLines().clear();
    textFile.getLines().add("testing,123,testing,123");
    textFile.save();
}
```

This was my initial data:

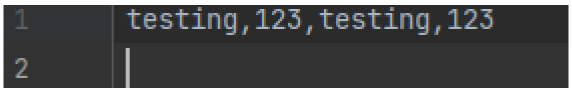


```
1    hello,world,my
2    name,is,coder!
```

And this was the result of the test function:



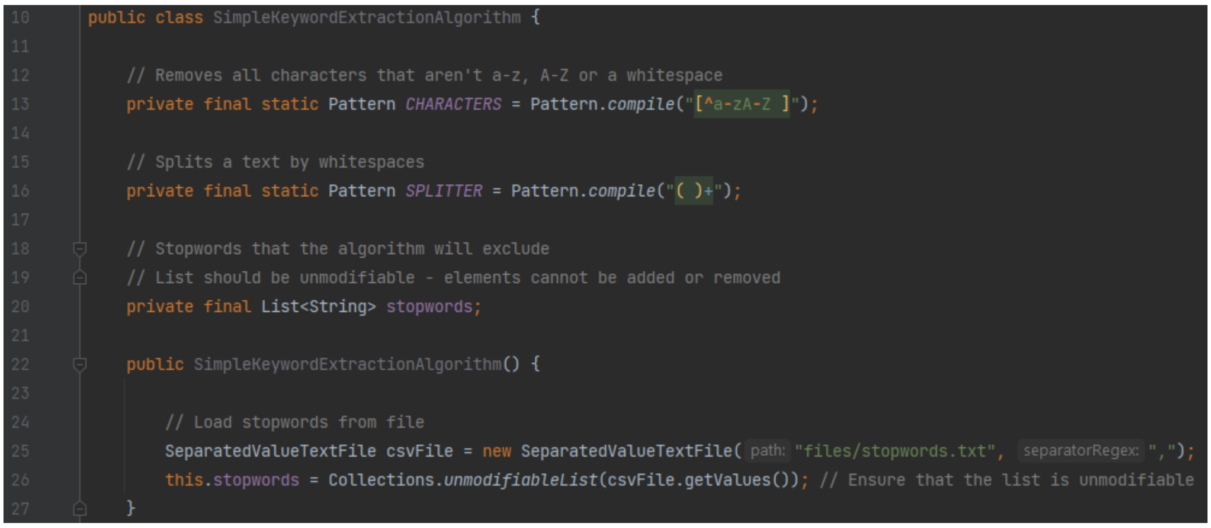
```
"C:\Program Files\Java\jdk1.8.0_301
hello
world
my
name
is
coder!

Process finished with exit code 0
```



```
1    testing,123,testing,123
2
```

As can be seen, the class works as intended as data is read successfully, split up by the separator value passed and wrote successfully too.

Finally, I can start writing my simple keyword extraction algorithm code. I start by coding the constructor to load the stopwords. The stopwords link to requirement 2.8.

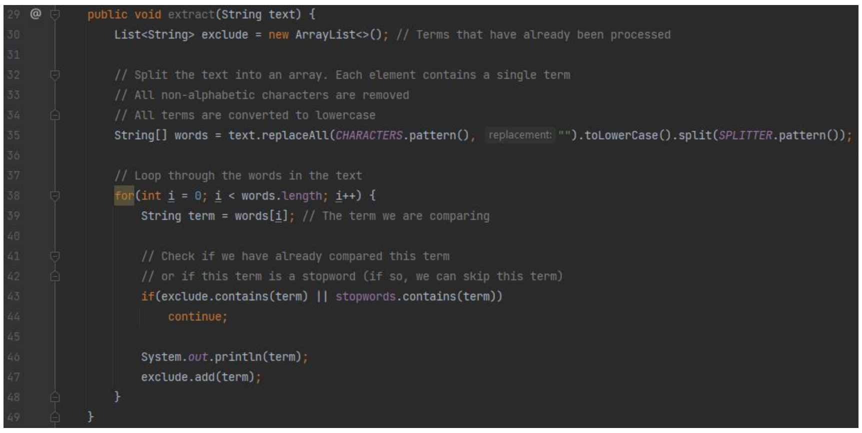


```java
public class SimpleKeywordExtractionAlgorithm {

    // Removes all characters that aren't a-z, A-Z or a whitespace
    private final static Pattern CHARACTERS = Pattern.compile("[^a-zA-Z ]");

    // Splits a text by whitespaces
    private final static Pattern SPLITTER = Pattern.compile("( )+");

    // Stopwords that the algorithm will exclude
    // List should be unmodifiable - elements cannot be added or removed
    private final List<String> stopwords;

    public SimpleKeywordExtractionAlgorithm() {

        // Load stopwords from file
        SeparatedValueTextFile csvFile = new SeparatedValueTextFile( path: "files/stopwords.txt",  separatorRegex: ",");
        this.stopwords = Collections.unmodifiableList(csvFile.getValues()); // Ensure that the list is unmodifiable
    }
```

The class has two static variables: CHARACTERS and SPLITTER. Static variables in Java are global and are shared amongst all objects of the class. Therefore, less memory space is used, and the variables only need to be allocated once.

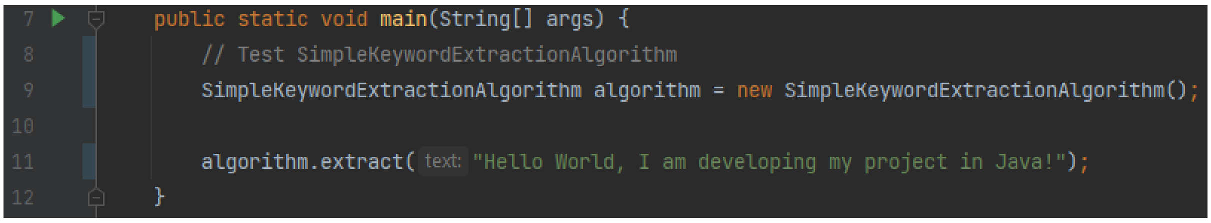
The next variable is the stopwords array. The array is initialised in the constructor and is a final variable. This means that it cannot be assigned to a different value during runtime. Therefore, we can assert that the value of the variable will never be null and therefore do not have to worry about checking for this in our algorithm. Furthermore, the list is unmodifiable. This means that elements cannot be added or removed during runtime. They can only be added when the list is first created. This ensures that no mistakes are made during coding, where a mistake might be removing stopwords from the list which would mess with the terms and produce inaccurate interest recommendations.

Next, I can start coding the extraction algorithm itself, the extract() function. To start, I split the text into individual terms. Next, I loop through each term. If the term has already been counted or the term is a stop word, we can continue to the next element in the loop. Otherwise, for now the algorithm simply prints the term to console.



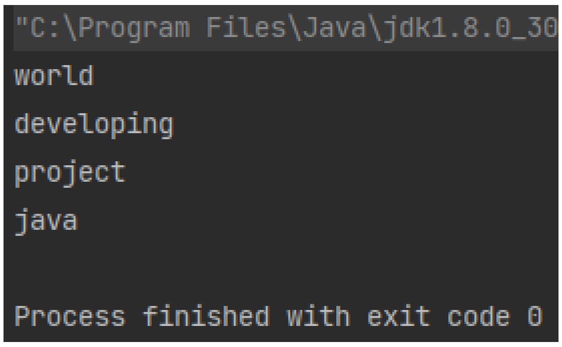
```java
@   public void extract(String text) {
        List<String> exclude = new ArrayList<>(); // Terms that have already been processed

        // Split the text into an array. Each element contains a single term
        // All non-alphabetic characters are removed
        // All terms are converted to lowercase
        String[] words = text.replaceAll(CHARACTERS.pattern(),  replacement: "").toLowerCase().split(SPLITTER.pattern());

        // Loop through the words in the text
        for(int i = 0; i < words.length; i++) {
            String term = words[i]; // The term we are comparing

            // Check if we have already compared this term
            // or if this term is a stopword (if so, we can skip this term)
            if(exclude.contains(term) || stopwords.contains(term))
                continue;

            System.out.println(term);
            exclude.add(term);
        }
    }
```

Running this test code and using the stopwords ‘hello,i,am,my,in’:



```java
public static void main(String[] args) {
    // Test SimpleKeywordExtractionAlgorithm
    SimpleKeywordExtractionAlgorithm algorithm = new SimpleKeywordExtractionAlgorithm();

    algorithm.extract( text: "Hello World, I am developing my project in Java!");
}
```

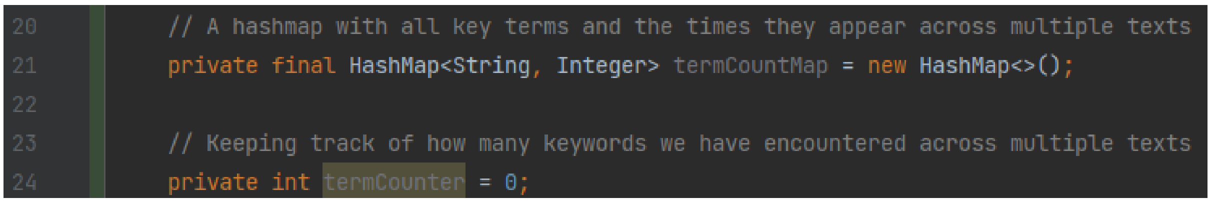
The output from the algorithm is as follows:



```
"C:\Program Files\Java\jdk1.8.0_30
world
developing
project
java

Process finished with exit code 0
```

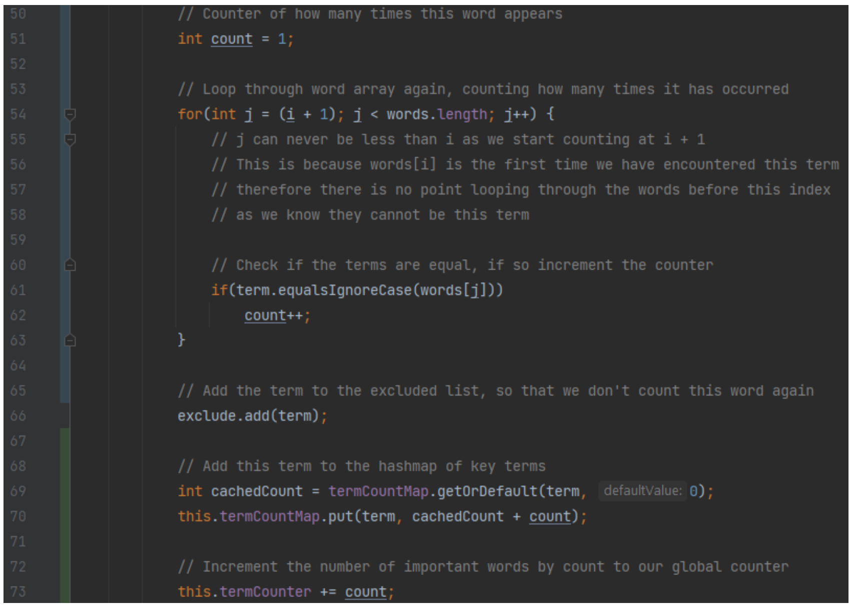
As it can be seen, the algorithm currently is splitting the text into its terms, removing any non-alphabetic characters, and removing any stop words.

Next, I created two new variables. A hash table/map linking strings (terms) to integers (occurrences within texts) and an integer tracking the total terms processed in the texts.



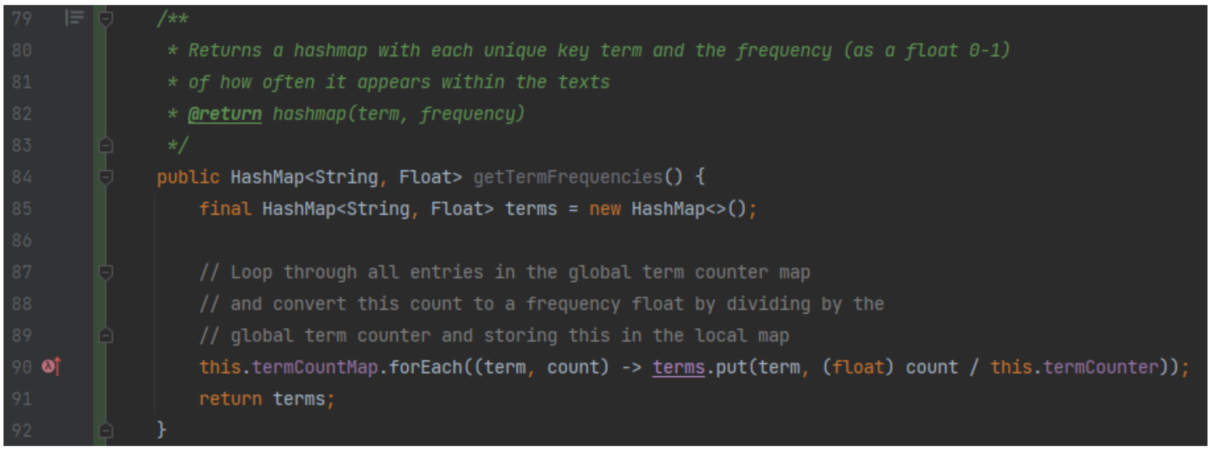
```java
// A hashmap with all key terms and the times they appear across multiple texts
private final HashMap<String, Integer> termCountMap = new HashMap<>();

// Keeping track of how many keywords we have encountered across multiple texts
private int termCounter = 0;
```

These variables are used to track the terms and their appearances and will determine the most important words from a text by their occurrences. Adjusting the extract() algorithm to the following, the implementation is finalised:



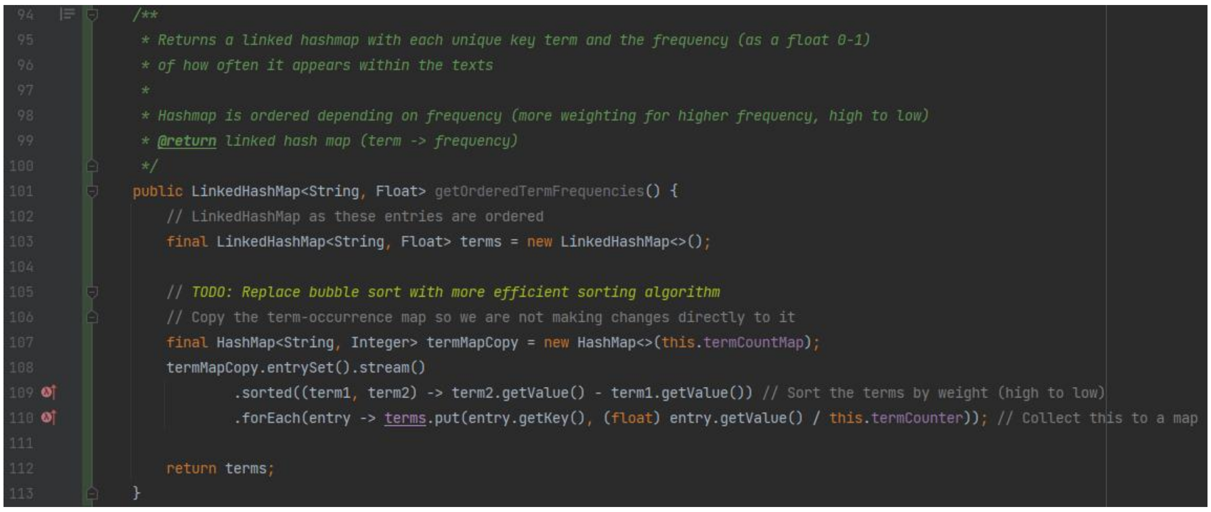
```java
// Counter of how many times this word appears
int count = 1;

// Loop through word array again, counting how many times it has occurred
for(int j = (i + 1); j < words.length; j++) {
    // j can never be less than i as we start counting at i + 1
    // This is because words[i] is the first time we have encountered this term
    // therefore there is no point looping through the words before this index
    // as we know they cannot be this term

    // Check if the terms are equal, if so increment the counter
    if(term.equalsIgnoreCase(words[j]))
        count++;
}

// Add the term to the excluded list, so that we don't count this word again
exclude.add(term);

// Add this term to the hashmap of key terms
int cachedCount = termCountMap.getOrDefault(term,  defaultValue: 0);
this.termCountMap.put(term, cachedCount + count);

// Increment the number of important words by count to our global counter
this.termCounter += count;
```

Lastly, to this class I need two final functions. The first function converts the term-occurrence map into a term-frequency map. This is done by dividing the occurrences of the term by the total term count.



```java
/**
 * Returns a hashmap with each unique key term and the frequency (as a float 0-1)
 * of how often it appears within the texts
 * @return hashmap(term, frequency)
 */
public HashMap<String, Float> getTermFrequencies() {
    final HashMap<String, Float> terms = new HashMap<>();

    // Loop through all entries in the global term counter map
    // and convert this count to a frequency float by dividing by the
    // global term counter and storing this in the local map
    this.termCountMap.forEach((term, count) -> terms.put(term, (float) count / this.termCounter));
    return terms;
}
```

The next method will simply order this term-frequency map. For now, I will implement this using a bubble sort, as this is the simplest and quickest to code and will allow me to carry out testing and develop other parts of my program that take higher priority than an efficient sorting algorithm. However, I will add a TODO note that my IDE will remind me to come back to this algorithm later to implement a more efficient sorting algorithm, such as a quick or merge sort.



```java
/**
 * Returns a linked hashmap with each unique key term and the frequency (as a float 0-1)
 * of how often it appears within the texts
 *
 * Hashmap is ordered depending on frequency (more weighting for higher frequency, high to low)
 * @return linked hash map (term -> frequency)
 */
public LinkedHashMap<String, Float> getOrderedTermFrequencies() {
    // LinkedHashMap as these entries are ordered
    final LinkedHashMap<String, Float> terms = new LinkedHashMap<>();

    // TODO: Replace bubble sort with more efficient sorting algorithm
    // Copy the term-occurrence map so we are not making changes directly to it
    final HashMap<String, Integer> termMapCopy = new HashMap<>(this.termCountMap);
    termMapCopy.entrySet().stream()
            .sorted((term1, term2) -> term2.getValue() - term1.getValue()) // Sort the terms by weight (high to low)
            .forEach(entry -> terms.put(entry.getKey(), (float) entry.getValue() / this.termCounter)); // Collect this to a map

    return terms;
}
```

Review

This iteration was very successful. I did not encounter any issues, bugs or errors in my code while testing which is a very positive outcome and saved on a lot of development time.

From this first iteration of the recommendation module, I will mark requirements that have been partially met in yellow/amber or fully completed in green. While only one success criteria were met and a second was partially met, a lot of backbone coding was done. This is very much reusable code and will speed up development further down the line.

Requirement Number	Requirement	Justification	Success Criteria & Evidence	Provided Evidence
2. Recommendation module				
2.1	Rectangular boxes stacked vertically	Text is read left-to-right and therefore is more readable	Screenshot of correct layout	
2.2	Boxes expand upon being clicked	To give more information on a particular account, the box will expand once clicked to display this	Box must show more information when the user clicks it. Screenshot of expanded information view	
2.3	Boxes display information about recommendations	Information such as the similarities in interests, account name, etc will allow the user to decide whether to follow	Screenshots of information being displayed	
2.4	List of interests based on user	To give recommend-dations, a list of interests is needed about the account, which is gained via the Keyword Extraction Algorithm	The algorithm is given a text and will return a list of strings that are the important parts of the text. Screenshots of the input and output texts	Simple extraction algorithm currently removes stopwords and only returns a list of important terms in a text. Screenshots were provided
2.5	Follow button	Saves time for the user as they can quickly follow the account from the program	Once the button is pressed, the user will now be following the account on Twitter. Screenshots of before vs after button press	
2.6	Ignore button	Allows the user to block the account if they do not want to see it again	Once the account is blocked, they will never be recommended to the user again	
2.7	Interest search bar	Allows the user to search for a particular	Once searched, the user will receive recommend-	

		subject/topic they are interested in	dations based on that interest. Screenshots of relevant accounts being displayed	
🟩 2.8	Stop words	Stop words are words that will be excluded by the algorithm and so will save processing time as non-important words will not be processed	Algorithm does not output any stop words with pictures to demonstrate this working	The algorithm removes words that are listed in the stopwords text file. Screenshots were provided and tests carried out to prove this
2.9	Match languages	Only processes texts/tweets that are English (including US English) to match the stakeholder's language and reduce inaccurate recommendations	Algorithm should exclude processing texts from users that are not English speaking	

Iteration three – generating recommendations

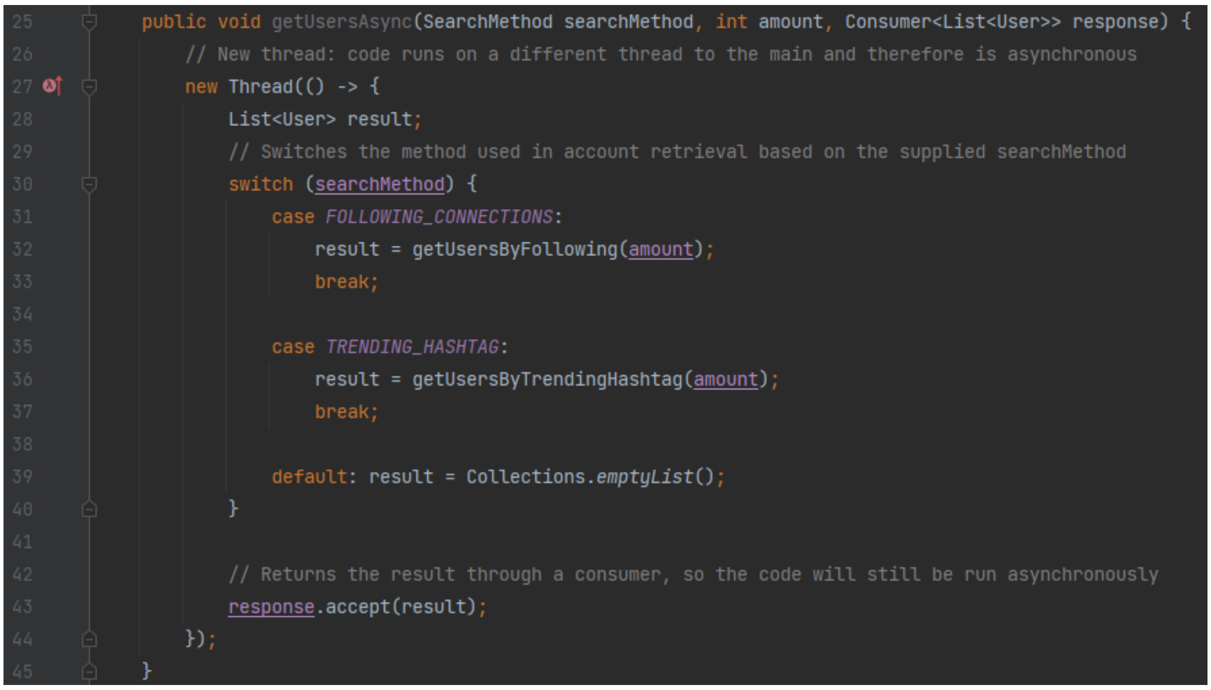
Now that a lot of backbone code has been developed, the time taken from now on to complete the rest of this section should be reduced as this code is modular and reusable. After reviewing, it is clear what still needs to be completed and what must wait for a further iteration – once the GUI is implemented.

Firstly, to recommend new accounts to follow, we must have a list of accounts. This is why the first step is discovery. This can be done in a variety of ways:

1. Breadth/depth first search on the user's following/followers. This is guaranteed to return a list of accounts with mutual connections to the user.
2. Pick a random trending hashtag, filter by accounts that have received a fair number of interactions on their tweets/posts under that hashtag and return a list of these. These accounts are more often than not going to be hot topics, popular, interesting, etc…

As both of these methods require many API calls (to retrieve Twitter accounts), I will be coding this asynchronously. The code will be executed on a different thread from the main thread which the rest of the program will be running on, such as the GUI. This will avoid blocking the main thread and will avoid causing my program to become unresponsive, improving useability for the end user as the program will remain responsive at all times.

I implemented this using this code:

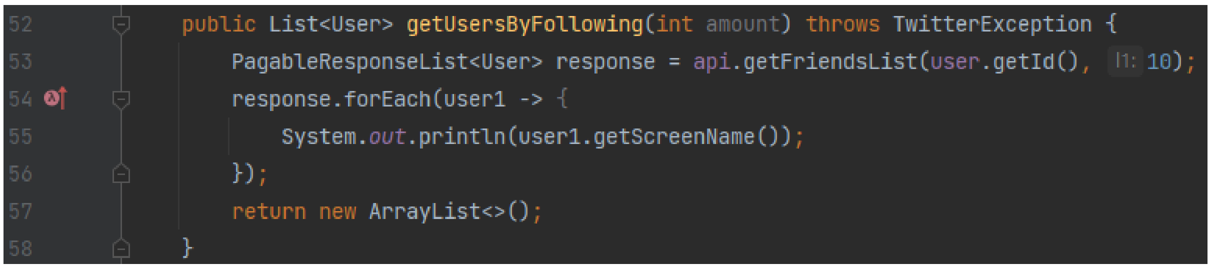


```java
public void getUsersAsync(SearchMethod searchMethod, int amount, Consumer<List<User>> response) {
    // New thread: code runs on a different thread to the main and therefore is asynchronous
    new Thread(() -> {
        List<User> result;
        // Switches the method used in account retrieval based on the supplied searchMethod
        switch (searchMethod) {
            case FOLLOWING_CONNECTIONS:
                result = getUsersByFollowing(amount);
                break;

            case TRENDING_HASHTAG:
                result = getUsersByTrendingHashtag(amount);
                break;

            default: result = Collections.emptyList();
        }

        // Returns the result through a consumer, so the code will still be run asynchronously
        response.accept(result);
    });
}
```

As the screenshot shows, the code is run on a different thread and therefore runs no risk of blocking the main thread while interacting with the Twitter API and waiting for web responses.

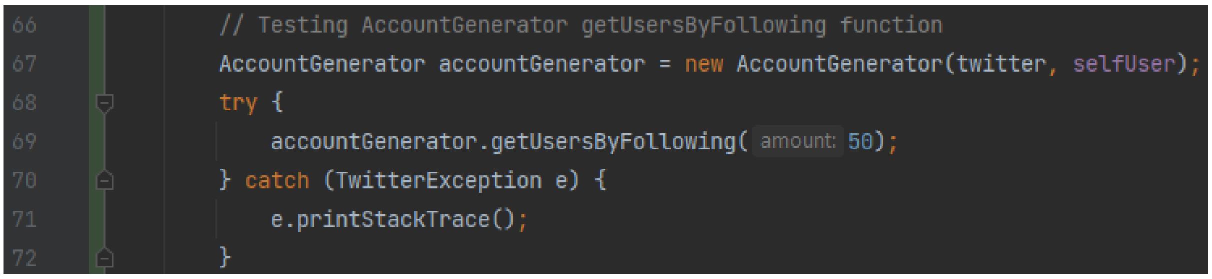
The switch statement allows for easy code maintainability as in the future additional search methods can be coded in easily by adding another case. Compared to using if-else blocks which would get confusing quickly with a larger number of cases. Each case in the switch statement calls a different function to return a list of accounts based on the desired method. These functions will also run asynchronously to the main thread as they are being called on the new thread.

The first search method I will make is the by followers/following. I made this code to test the response that the Twitter API gives you when using 'api.getFriendsList(user id, amount)'. As I had not coded with this method before, I created a test to see what the result of it is.

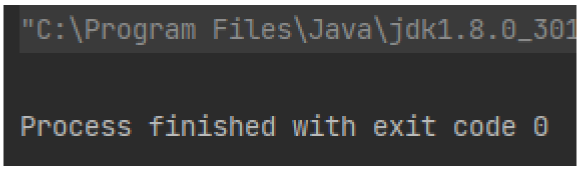


```java
public List<User> getUsersByFollowing(int amount) throws TwitterException {
    PagableResponseList<User> response = api.getFriendsList(user.getId(), l1: 10);
    response.forEach(user1 -> {
        System.out.println(user1.getScreenName());
    });
    return new ArrayList<>();
}
```

And using this test code:



```java
// Testing AccountGenerator getUsersByFollowing function
AccountGenerator accountGenerator = new AccountGenerator(twitter, selfUser);
try {
    accountGenerator.getUsersByFollowing( amount: 50);
} catch (TwitterException e) {
    e.printStackTrace();
}
```

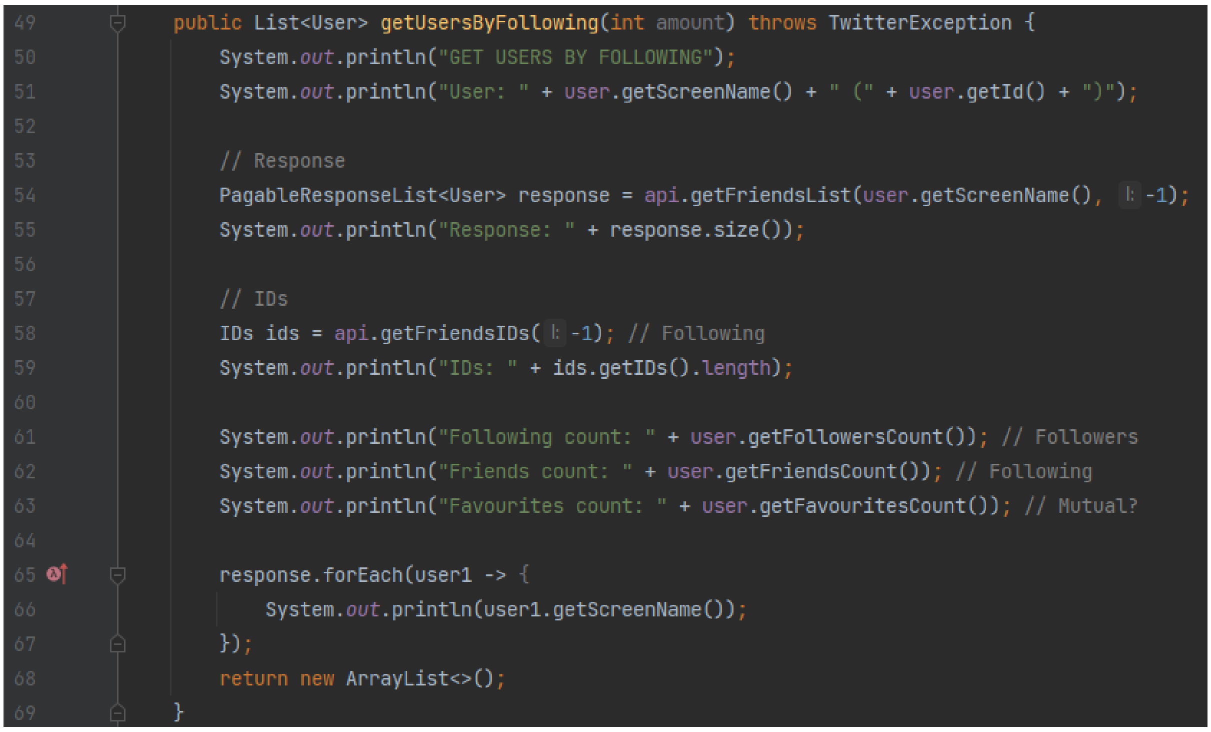


```
"C:\Program Files\Java\jdk1.8.0_301

Process finished with exit code 0
```

And when I ran it, there was no response. (As pictured to the right)

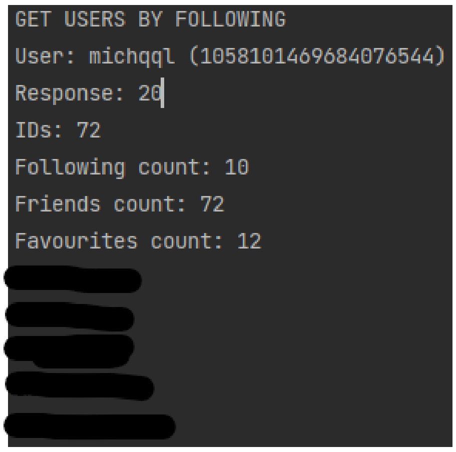
So now, I will add a few debug statements into my code to check if there are any points in my code that are not being reached.

After testing code, writing more debug statements, and running more test code once again, I finally resolved the problem. My final test/debug function looked like so:



```java
    public List<User> getUsersByFollowing(int amount) throws TwitterException {
        System.out.println("GET USERS BY FOLLOWING");
        System.out.println("User: " + user.getScreenName() + " (" + user.getId() + ")");

        // Response
        PagableResponseList<User> response = api.getFriendsList(user.getScreenName(), l: -1);
        System.out.println("Response: " + response.size());

        // IDs
        IDs ids = api.getFriendsIDs( l: -1); // Following
        System.out.println("IDs: " + ids.getIDs().length);

        System.out.println("Following count: " + user.getFollowersCount()); // Followers
        System.out.println("Friends count: " + user.getFriendsCount()); // Following
        System.out.println("Favourites count: " + user.getFavouritesCount()); // Mutual?

        response.forEach(user1 -> {
            System.out.println(user1.getScreenName());
        });
        return new ArrayList<>();
    }
```

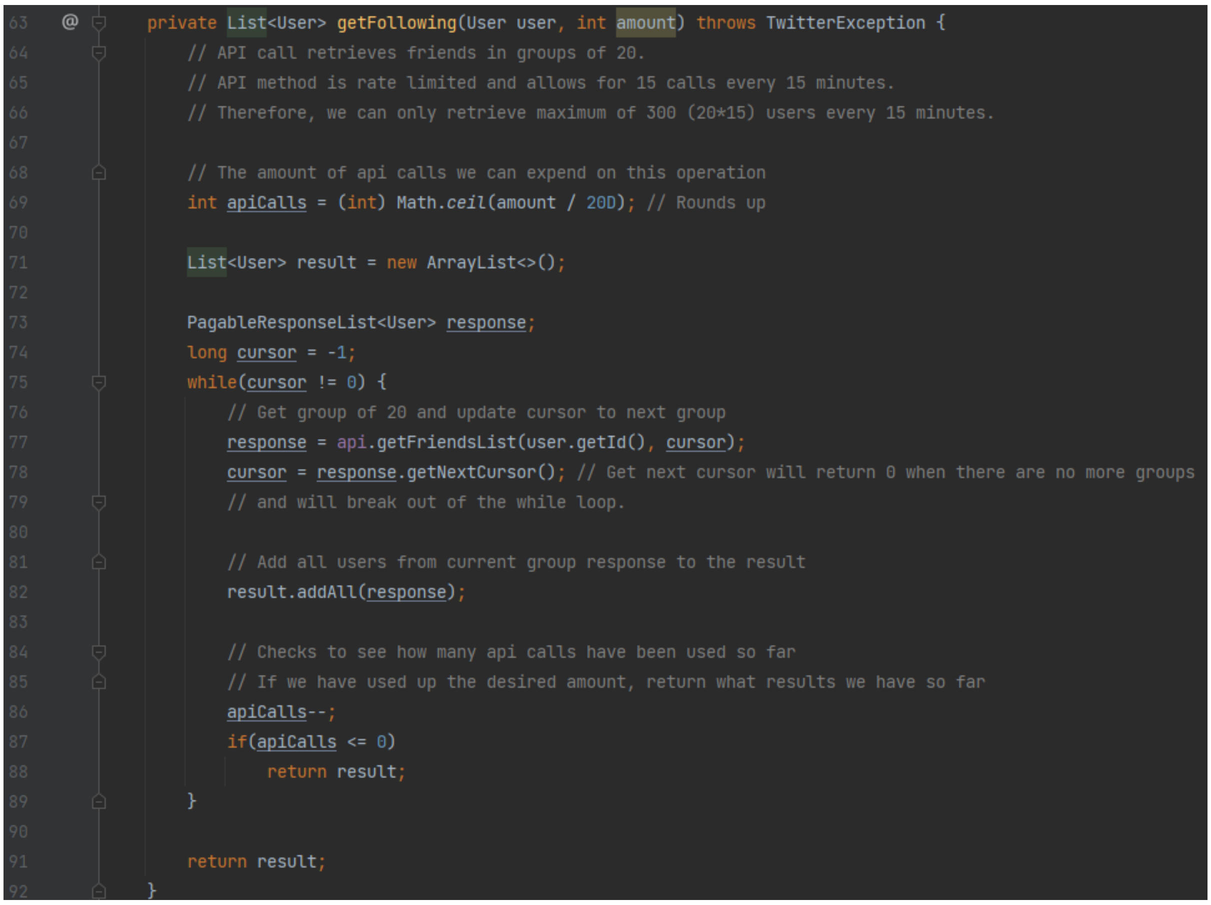
The issue occurred at 'api.getFriendsList(user id, number)'. I had put number to 10, which did not work. However, when I set it to -1 it worked perfectly. In console, the debug statements proved it is now working.



```
GET USERS BY FOLLOWING
User: michqql (1058101469684076544)
Response: 20
IDs: 72
Following count: 10
Friends count: 72
Favourites count: 12
[redacted]
```

I took these actions (debug statements and rerunning code) as I knew that the issue was lying in this method as it was not outputting anything to console as it should have been. Therefore, it made sense to add more and more debug statements exhaustively testing to pinpoint the solution. In the end, it was only a small change to a single number. The number (10 instead of -1) was causing the issue as in the api to number corresponds to a 'cursor pointer' named by Twitter. In twitter, a cursor is an index/pointer to a resource in time. -1 indicates to get a new list, whereas 10 was trying to retrieve a list at index 10 which did not exist.

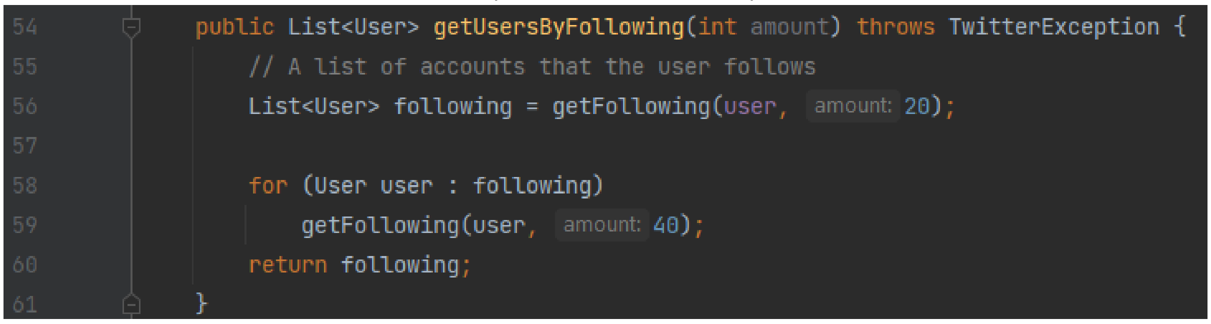
To retrieve a list of accounts, I coded this



```java
private List<User> getFollowing(User user, int amount) throws TwitterException {
    // API call retrieves friends in groups of 20.
    // API method is rate limited and allows for 15 calls every 15 minutes.
    // Therefore, we can only retrieve maximum of 300 (20*15) users every 15 minutes.

    // The amount of api calls we can expend on this operation
    int apiCalls = (int) Math.ceil(amount / 20D); // Rounds up

    List<User> result = new ArrayList<>();

    PagableResponseList<User> response;
    long cursor = -1;
    while(cursor != 0) {
        // Get group of 20 and update cursor to next group
        response = api.getFriendsList(user.getId(), cursor);
        cursor = response.getNextCursor(); // Get next cursor will return 0 when there are no more groups
        // and will break out of the while loop.

        // Add all users from current group response to the result
        result.addAll(response);

        // Checks to see how many api calls have been used so far
        // If we have used up the desired amount, return what results we have so far
        apiCalls--;
        if(apiCalls <= 0)
            return result;
    }

    return result;
}
```

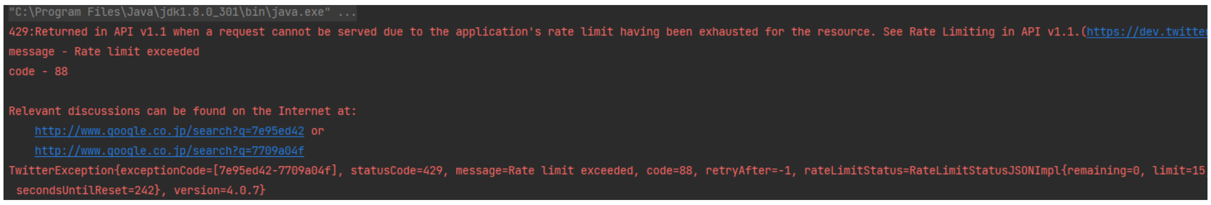
The algorithm considers how many api calls it should make, as to avoid rate limitations by Twitter.

When testing, Twitter does in fact limit to 15 requests per 15 minutes.
The code below makes 1 + 20 x 2 = 41 requests in a short time period of a few milliseconds.



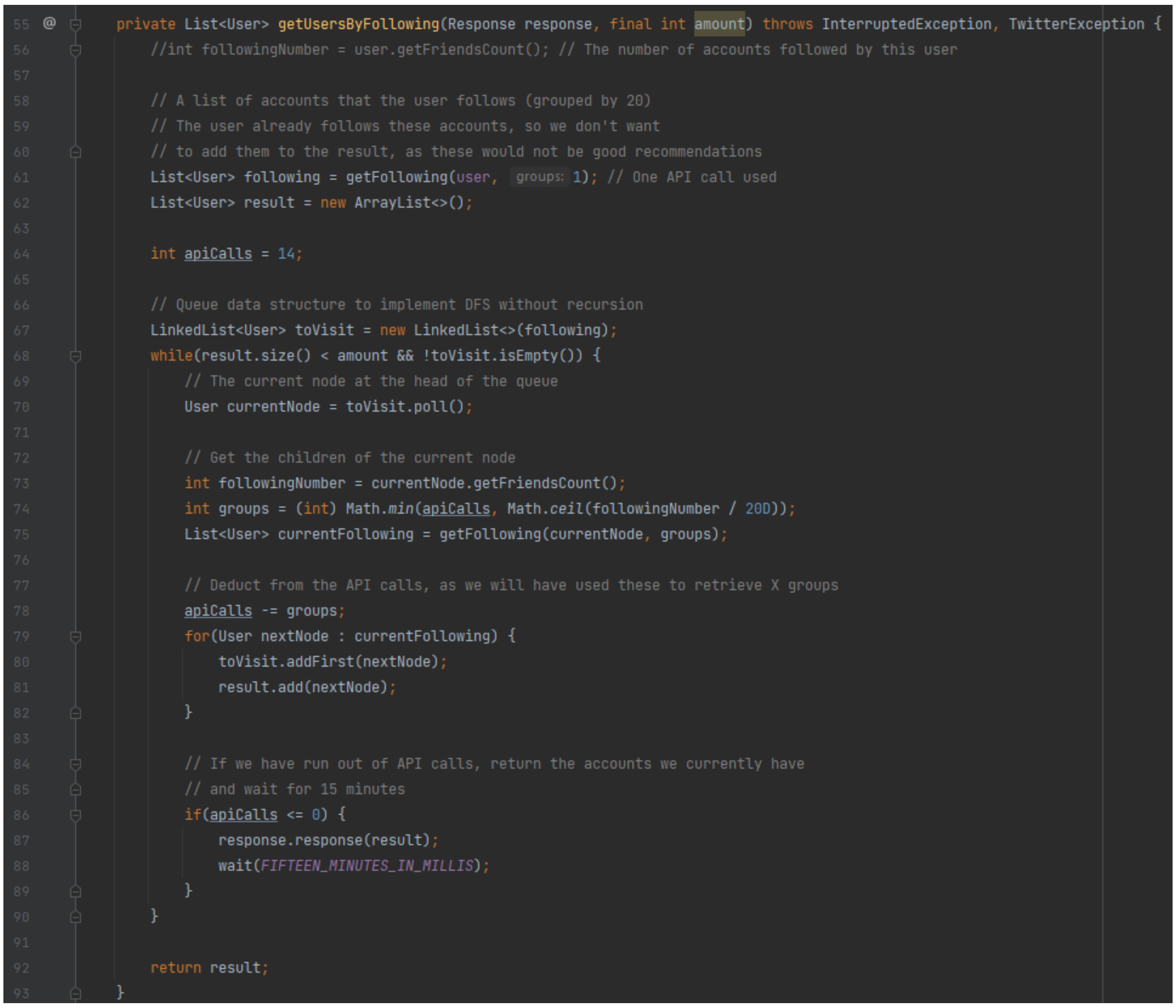
```java
public List<User> getUsersByFollowing(int amount) throws TwitterException {
    // A list of accounts that the user follows
    List<User> following = getFollowing(user, amount: 20);

    for (User user : following)
        getFollowing(user, amount: 40);
    return following;
}
```

The result is therefore getting rate limited as expected. As can be seen, it states in the error message that the limit is 15 and seconds until reset are 242.



```
"C:\Program Files\Java\jdk1.8.0_301\bin\java.exe" ...
429:Returned in API v1.1 when a request cannot be served due to the application's rate limit having been exhausted for the resource. See Rate Limiting in API v1.1.(https://dev.twitte
message - Rate limit exceeded
code - 88

Relevant discussions can be found on the Internet at:
    http://www.google.co.jp/search?q=7e95ed42 or
    http://www.google.co.jp/search?q=7709a04f
TwitterException{exceptionCode=[7e95ed42-7709a04f], statusCode=429, message=Rate limit exceeded, code=88, retryAfter=-1, rateLimitStatus=RateLimitStatusJSONImpl{remaining=0, limit=15
 secondsUntilReset=242}, version=4.0.7}
```

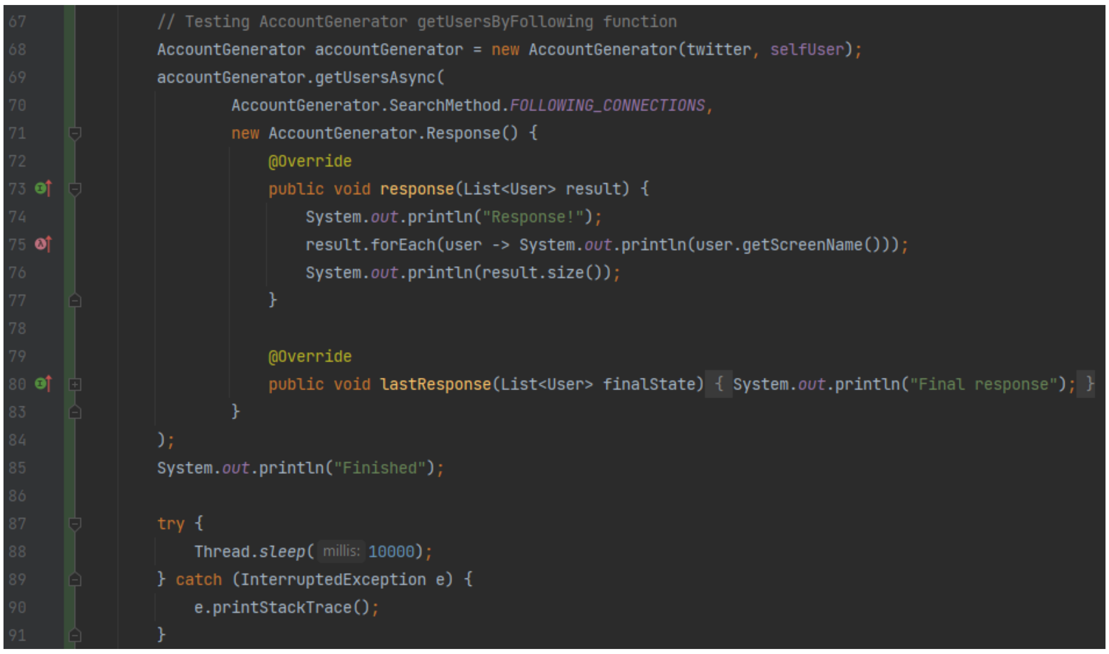
To overcome this rate limitation and ensure that no errors are thrown which could crash the program, I am using a secondary thread (as mentioned before – asynchronous) and returning results momentarily. As this method was designed to use a depth-first search, I have implemented this using a queue and iteration instead of recursion. This is to save on memory as it does not use a call stack and is easier to manage in my opinion as all the variables can be modified in-situ.



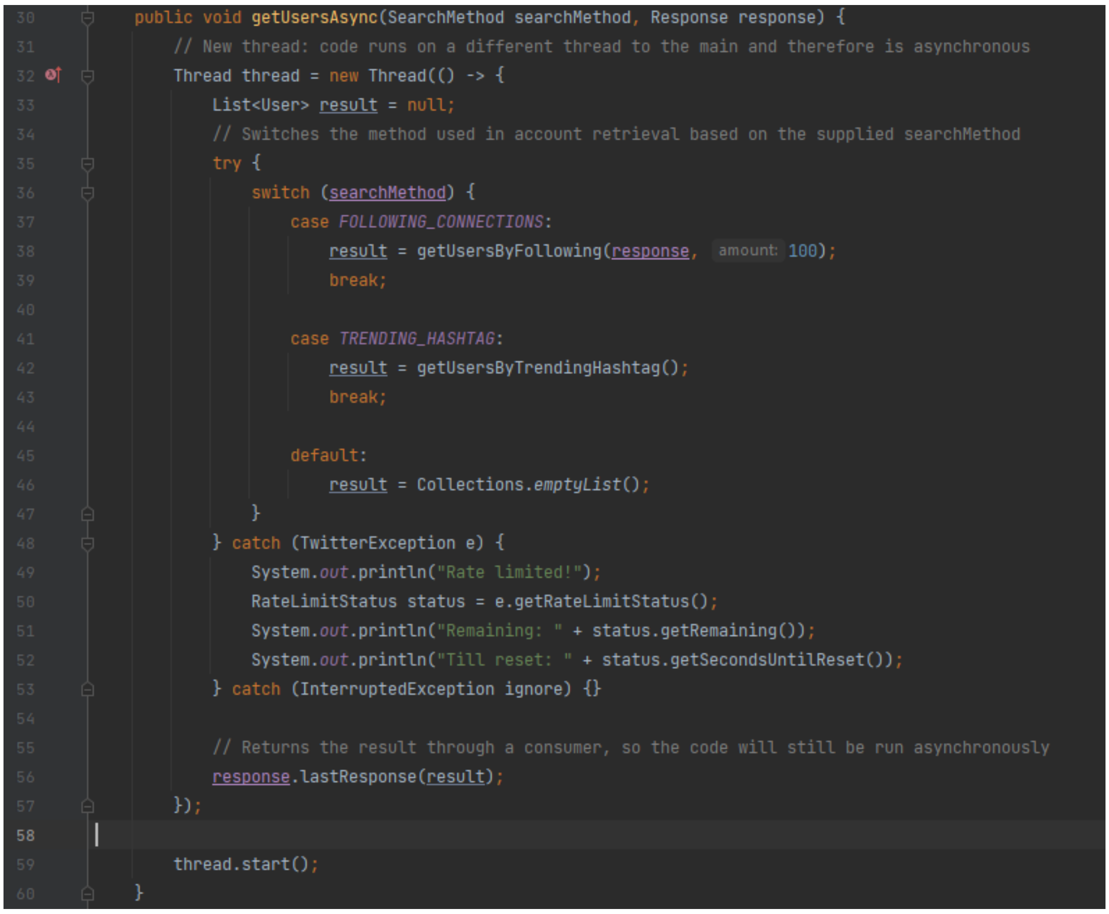
```java
private List<User> getUsersByFollowing(Response response, final int amount) throws InterruptedException, TwitterException {
    //int followingNumber = user.getFriendsCount(); // The number of accounts followed by this user

    // A list of accounts that the user follows (grouped by 20)
    // The user already follows these accounts, so we don't want
    // to add them to the result, as these would not be good recommendations
    List<User> following = getFollowing(user, groups: 1); // One API call used
    List<User> result = new ArrayList<>();

    int apiCalls = 14;

    // Queue data structure to implement DFS without recursion
    LinkedList<User> toVisit = new LinkedList<>(following);
    while(result.size() < amount && !toVisit.isEmpty()) {
        // The current node at the head of the queue
        User currentNode = toVisit.poll();

        // Get the children of the current node
        int followingNumber = currentNode.getFriendsCount();
        int groups = (int) Math.min(apiCalls, Math.ceil(followingNumber / 20D));
        List<User> currentFollowing = getFollowing(currentNode, groups);

        // Deduct from the API calls, as we will have used these to retrieve X groups
        apiCalls -= groups;
        for(User nextNode : currentFollowing) {
            toVisit.addFirst(nextNode);
            result.add(nextNode);
        }

        // If we have run out of API calls, return the accounts we currently have
        // and wait for 15 minutes
        if(apiCalls <= 0) {
            response.response(result);
            wait(FIFTEEN_MINUTES_IN_MILLIS);
        }
    }

    return result;
}
```

Currently, I am experiencing a problem with this code. As the code to get the accounts is running on a different thread, the output is being returned after the main thread has finished execution. This is a problem as the main thread finishes execution all other threads are terminated as the program halts. This makes it difficult to debug whether my program is working correctly. I am using many debug statements but after a while of testing, I found that telling the main thread to sleep has solved my problem for now. As the main thread sleeps, the other thread is able to carry out the function and return a result. This will not be a problem when I implement the GUI, as the GUI is an example of event-driven architecture, the program will not halt as the GUI is open and is waiting for input from the user. Therefore, once the GUI is implemented and while the program is running and waiting for input, the asynchronous thread will be able to carry out its execution fine and return a result.

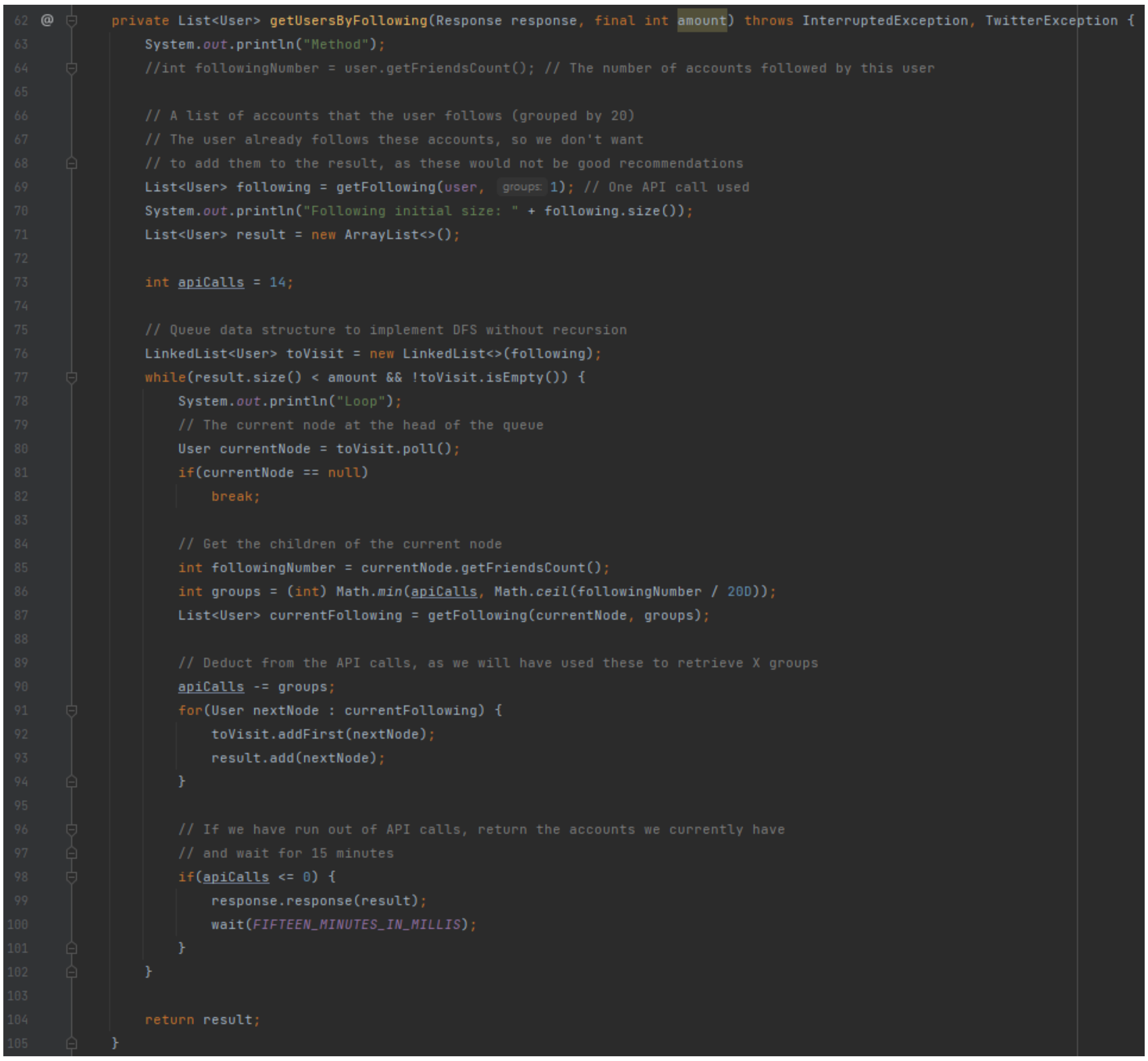
Here shows the code I used to test:



```java
        // Testing AccountGenerator getUsersByFollowing function
        AccountGenerator accountGenerator = new AccountGenerator(twitter, selfUser);
        accountGenerator.getUsersAsync(
                AccountGenerator.SearchMethod.FOLLOWING_CONNECTIONS,
                new AccountGenerator.Response() {
                    @Override
                    public void response(List<User> result) {
                        System.out.println("Response!");
                        result.forEach(user -> System.out.println(user.getScreenName()));
                        System.out.println(result.size());
                    }

                    @Override
                    public void lastResponse(List<User> finalState) { System.out.println("Final response"); }
                }
        );
        System.out.println("Finished");

        try {
            Thread.sleep( millis: 10000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
```



```java
    public void getUsersAsync(SearchMethod searchMethod, Response response) {
        // New thread: code runs on a different thread to the main and therefore is asynchronous
        Thread thread = new Thread(() -> {
            List<User> result = null;
            // Switches the method used in account retrieval based on the supplied searchMethod
            try {
                switch (searchMethod) {
                    case FOLLOWING_CONNECTIONS:
                        result = getUsersByFollowing(response,  amount: 100);
                        break;

                    case TRENDING_HASHTAG:
                        result = getUsersByTrendingHashtag();
                        break;

                    default:
                        result = Collections.emptyList();
                }
            } catch (TwitterException e) {
                System.out.println("Rate limited!");
                RateLimitStatus status = e.getRateLimitStatus();
                System.out.println("Remaining: " + status.getRemaining());
                System.out.println("Till reset: " + status.getSecondsUntilReset());
            } catch (InterruptedException ignore) {}

            // Returns the result through a consumer, so the code will still be run asynchronously
            response.lastResponse(result);
        });

        thread.start();
    }
```



```java
    private List<User> getUsersByFollowing(Response response, final int amount) throws InterruptedException, TwitterException {
        System.out.println("Method");
        //int followingNumber = user.getFriendsCount(); // The number of accounts followed by this user

        // A list of accounts that the user follows (grouped by 20)
        // The user already follows these accounts, so we don't want
        // to add them to the result, as these would not be good recommendations
        List<User> following = getFollowing(user, groups: 1); // One API call used
        System.out.println("Following initial size: " + following.size());
        List<User> result = new ArrayList<>();

        int apiCalls = 14;

        // Queue data structure to implement DFS without recursion
        LinkedList<User> toVisit = new LinkedList<>(following);
        while(result.size() < amount && !toVisit.isEmpty()) {
            System.out.println("Loop");
            // The current node at the head of the queue
            User currentNode = toVisit.poll();
            if(currentNode == null)
                break;

            // Get the children of the current node
            int followingNumber = currentNode.getFriendsCount();
            int groups = (int) Math.min(apiCalls, Math.ceil(followingNumber / 200D));
            List<User> currentFollowing = getFollowing(currentNode, groups);

            // Deduct from the API calls, as we will have used these to retrieve X groups
            apiCalls -= groups;
            for(User nextNode : currentFollowing) {
                toVisit.addFirst(nextNode);
                result.add(nextNode);
            }

            // If we have run out of API calls, return the accounts we currently have
            // and wait for 15 minutes
            if(apiCalls <= 0) {
                response.response(result);
                wait(FIFTEEN_MINUTES_IN_MILLIS);
            }
        }

        return result;
    }
```

To test, I will use the following test cases to see the output to console

Test Case	Expected Outcome	Actual Outcome
No rate limiting from Twitter	A list of accounts is returned	Result as expected
API calls have been used (15 calls per 15 minutes)	Error is thrown as Twitter refuses to make any further API calls for a certain time period	Result as expected

On second thought, to overcome this rate limitation (currently, a maximum of 300 accounts can be retrieved every 15 minutes) I can use a different API call. This time, instead of it returning the entire account object, it only returns the account ID. This has less data and may require subsequent API calls down the line (at a higher rate of 900/15mins), however this will solve the rate limitation problem, as I will be able to retrieve 3500 account IDs per 15 minutes.

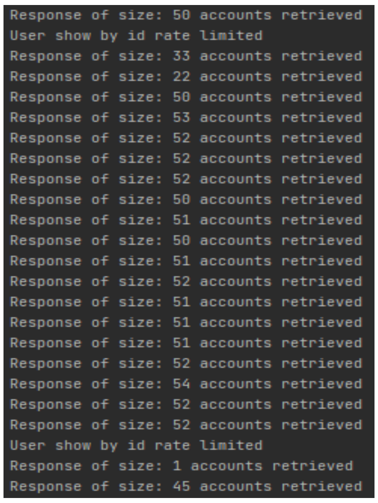
To make this change, I must first edit the 'getFollowing' function that I coded on page 21. The algorithm is now shown on the next page.

The algorithm starts by validating the current rate limit status. If there are zero remaining API calls in the current time window, the algorithm will tell the thread to sleep until these are reset. It then gets the user's friends. A timer is then started. At a regular interval the timer calls the main thread with a response of the current result of accounts retrieved. A depth-first search (DFS) is then used to get more accounts, which the user possibly may not follow. Again, rate limits are validated to ensure that api calls can be made, if not the thread is put back to sleep until it can work again. Once a threshold has been reached (specified in the code) the algorithm will stop and return its current contents. This is to ensure that the algorithm does not continue forever (or an indefinitely long amount of time) as there are an estimated 290 million Twitter accounts which the DFS would continue to traverse for a long time!

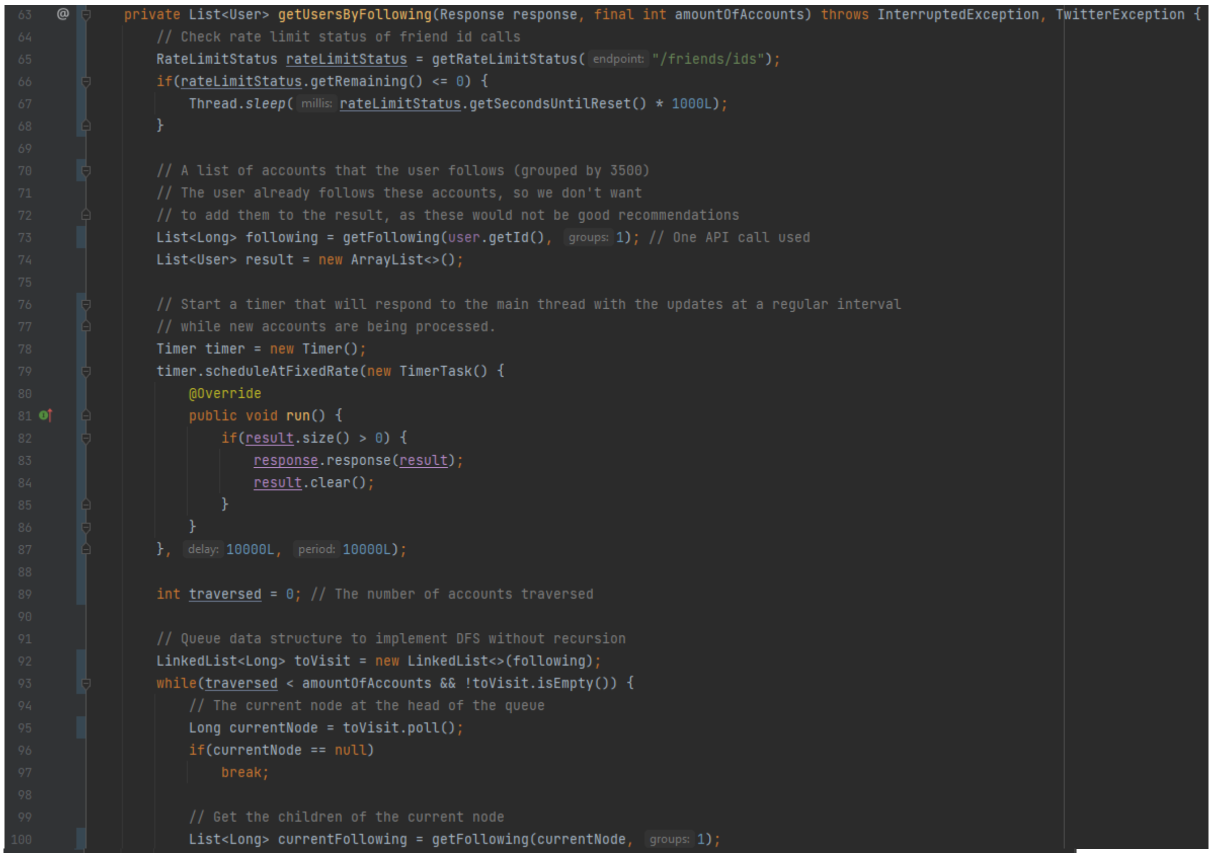
Testing and review

Test case: I will allow the algorithm to run for ~35 minutes.
Expected outcome: the algorithm will have exacerbated it's api calls two to three times. (Twice due to friend ID limit and once due to account retrieval)
Actual outcome:

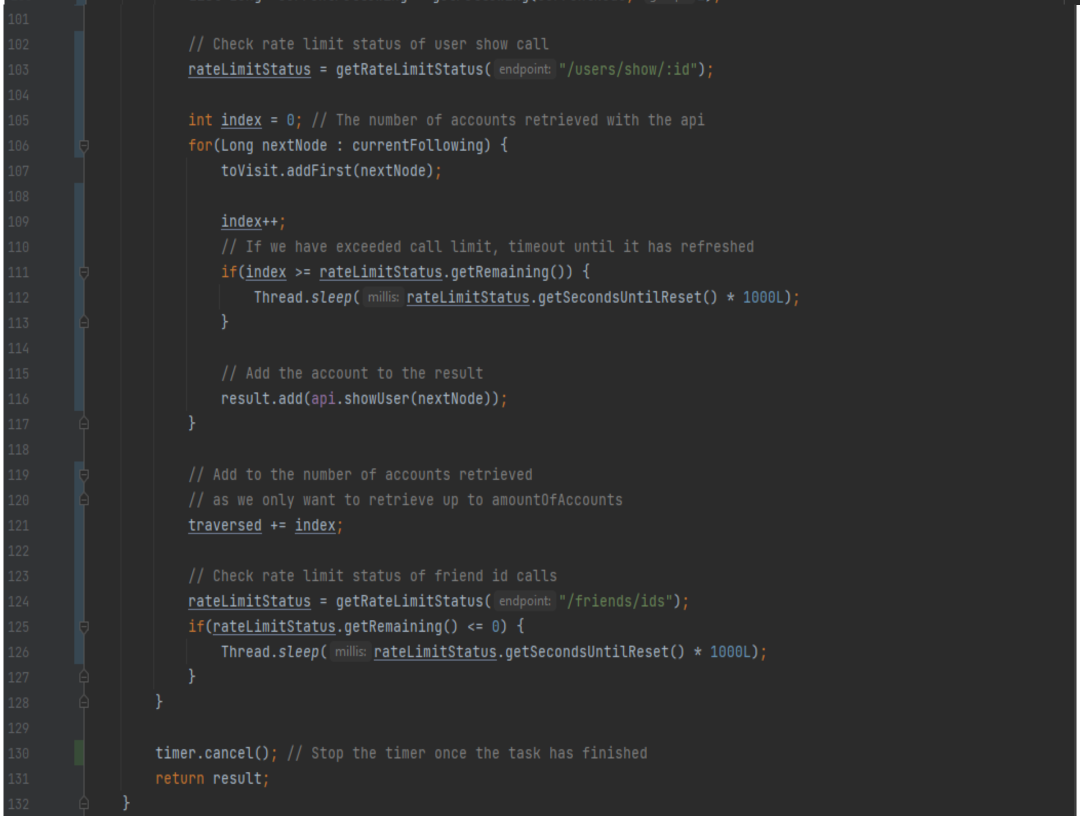
1. The algorithm finished execution after it reached the specified number of accounts to retrieve, as intended. To test for a longer period of time, I increased the desired number of accounts and re-ran the algorithm.
2. The algorithm ran for over 2 minutes until it crashed due to a rate limitation error by Twitter. I have added debug statements to find and fix the issue. I found the issue to be with the way I was handling rate limitations. I changed a few lines of code (detailed later on) and re-ran the algorithm.
3. After running successfully for a while, the program encountered a rate limit. As coded, the thread slept until Twitter reset the rate limits. However, when fetching the next user, an error was thrown stating there were 0 requests left and 0 seconds to reset! To avoid this, I added a 2 second delay before the program resumes to ensure the rate limit has reset. I will now re-run the algorithm.
4. The code lasted the full 35 minutes, with two pauses as expected. The result was as expected



```
Response of size: 50 accounts retrieved
User show by id rate limited
Response of size: 33 accounts retrieved
Response of size: 22 accounts retrieved
Response of size: 50 accounts retrieved
Response of size: 53 accounts retrieved
Response of size: 52 accounts retrieved
Response of size: 52 accounts retrieved
Response of size: 52 accounts retrieved
Response of size: 50 accounts retrieved
Response of size: 51 accounts retrieved
Response of size: 50 accounts retrieved
Response of size: 51 accounts retrieved
Response of size: 52 accounts retrieved
Response of size: 51 accounts retrieved
Response of size: 51 accounts retrieved
Response of size: 51 accounts retrieved
Response of size: 52 accounts retrieved
Response of size: 54 accounts retrieved
Response of size: 52 accounts retrieved
Response of size: 52 accounts retrieved
User show by id rate limited
Response of size: 1 accounts retrieved
Response of size: 45 accounts retrieved
```



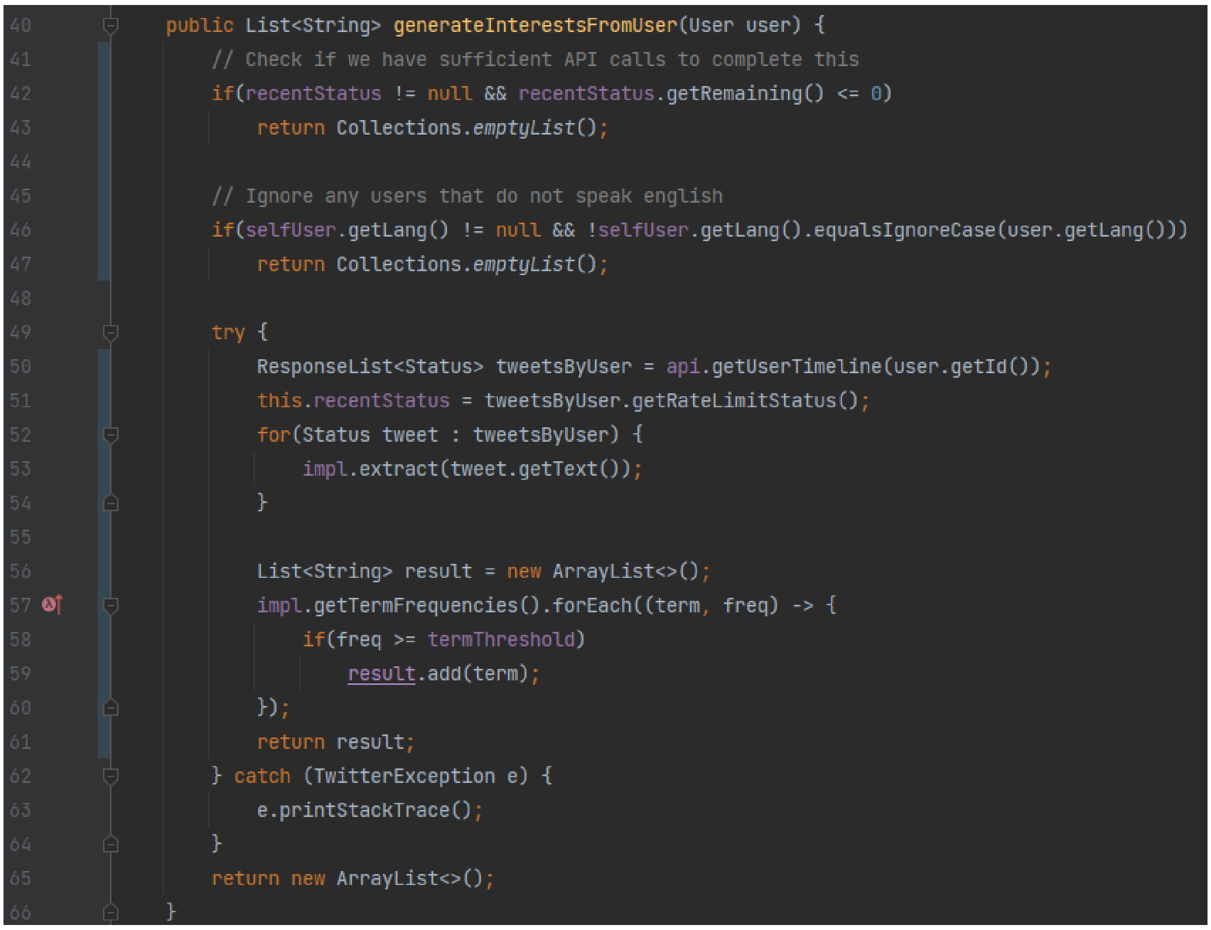
```
private List<User> getUsersByFollowing(Response response, final int amountOfAccounts) throws InterruptedException, TwitterException {
    // Check rate limit status of friend id calls
    RateLimitStatus rateLimitStatus = getRateLimitStatus( endpoint: "/friends/ids");
    if(rateLimitStatus.getRemaining() <= 0) {
        Thread.sleep( millis: rateLimitStatus.getSecondsUntilReset() * 1000L);
    }

    // A list of accounts that the user follows (grouped by 3500)
    // The user already follows these accounts, so we don't want
    // to add them to the result, as these would not be good recommendations
    List<Long> following = getFollowing(user.getId(),  groups: 1); // One API call used
    List<User> result = new ArrayList<>();

    // Start a timer that will respond to the main thread with the updates at a regular interval
    // while new accounts are being processed.
    Timer timer = new Timer();
    timer.scheduleAtFixedRate(new TimerTask() {
        @Override
        public void run() {
            if(result.size() > 0) {
                response.response(result);
                result.clear();
            }
        }
    },  delay: 10000L,  period: 10000L);

    int traversed = 0; // The number of accounts traversed

    // Queue data structure to implement DFS without recursion
    LinkedList<Long> toVisit = new LinkedList<>(following);
    while(traversed < amountOfAccounts && !toVisit.isEmpty()) {
        // The current node at the head of the queue
        Long currentNode = toVisit.poll();
        if(currentNode == null)
            break;

        // Get the children of the current node
        List<Long> currentFollowing = getFollowing(currentNode,  groups: 1);
```



```

        // Check rate limit status of user show call
        rateLimitStatus = getRateLimitStatus( endpoint: "/users/show/:id");

        int index = 0; // The number of accounts retrieved with the api
        for(Long nextNode : currentFollowing) {
            toVisit.addFirst(nextNode);

            index++;
            // If we have exceeded call limit, timeout until it has refreshed
            if(index >= rateLimitStatus.getRemaining()) {
                Thread.sleep( millis: rateLimitStatus.getSecondsUntilReset() * 1000L);
            }

            // Add the account to the result
            result.add(api.showUser(nextNode));
        }

        // Add to the number of accounts retrieved
        // as we only want to retrieve up to amountOfAccounts
        traversed += index;

        // Check rate limit status of friend id calls
        rateLimitStatus = getRateLimitStatus( endpoint: "/friends/ids");
        if(rateLimitStatus.getRemaining() <= 0) {
            Thread.sleep( millis: rateLimitStatus.getSecondsUntilReset() * 1000L);
        }
    }

    timer.cancel(); // Stop the timer once the task has finished
    return result;
}
```

Now I have coded an algorithm to get a list of accounts, I can get their tweets and pass these into my keyword extraction algorithm to get a list of interests per account.

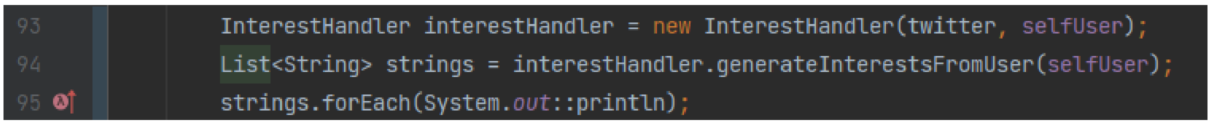
To get a list of interests from a user, I made this function:



```java
    public List<String> generateInterestsFromUser(User user) {
        // Check if we have sufficient API calls to complete this
        if(recentStatus != null && recentStatus.getRemaining() <= 0)
            return Collections.emptyList();

        // Ignore any users that do not speak english
        if(selfUser.getLang() != null && !selfUser.getLang().equalsIgnoreCase(user.getLang()))
            return Collections.emptyList();

        try {
            ResponseList<Status> tweetsByUser = api.getUserTimeline(user.getId());
            this.recentStatus = tweetsByUser.getRateLimitStatus();
            for(Status tweet : tweetsByUser) {
                impl.extract(tweet.getText());
            }

            List<String> result = new ArrayList<>();
            impl.getTermFrequencies().forEach((term, freq) -> {
                if(freq >= termThreshold)
                    result.add(term);
            });
            return result;
        } catch (TwitterException e) {
            e.printStackTrace();
        }
        return new ArrayList<>();
    }
```

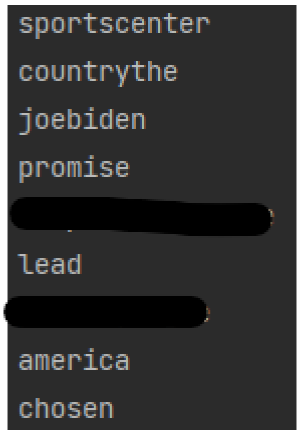
It uses the keyword extraction algorithm coded earlier (variable named 'impl' short for implementation as this algorithm is subject to change) to extract key terms from the tweets and adding these to the result if they are over a certain frequency threshold. Validation is used at the top of the function to ensure no errors will arise. For example, it validates that the program has not been rate limited first, then it checks if the language is the same as the users (as specified in requirement 2.9).

To test this has worked properly, I created the following test code to get the list of interests and print them to console.

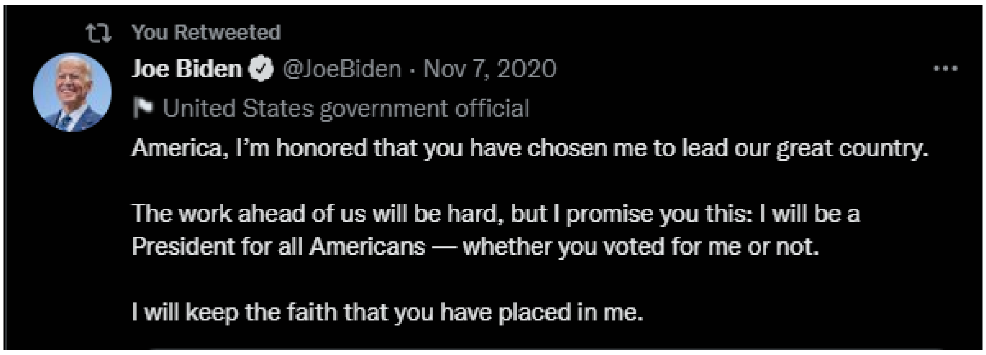


```java
        InterestHandler interestHandler = new InterestHandler(twitter, selfUser);
        List<String> strings = interestHandler.generateInterestsFromUser(selfUser);
        strings.forEach(System.out::println);
```

And after running the code, the output in console was the following (from the original text on right):



```
sportscenter
countrythe
joebiden
promise
[redacted]
lead
[redacted]
america
chosen
```



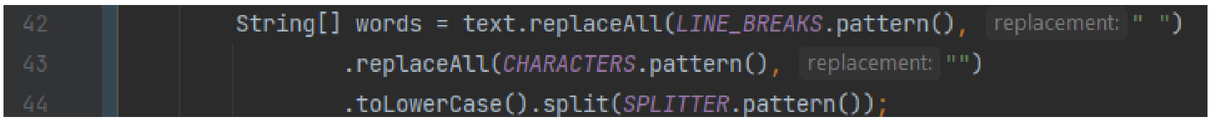
You Retweeted
Joe Biden @JoeBiden · Nov 7, 2020
United States government official
America, I'm honored that you have chosen me to lead our great country.

The work ahead of us will be hard, but I promise you this: I will be a President for all Americans — whether you voted for me or not.

I will keep the faith that you have placed in me.

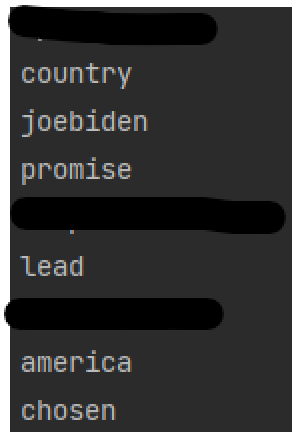
(Sportscenter and blanks were from another tweet. Countrythe is an issue that will be resolved)

To fix the issue with 'countrythe' which should clearly be two separate words as shown in the tweet, it is an issue to do with the new line character being removed when removing all non-alphabetic characters. To fix this, I made the following change in the keyword extraction algorithm:



```
        String[] words = text.replaceAll(LINE_BREAKS.pattern(), replacement: " ")
                .replaceAll(CHARACTERS.pattern(), replacement: "")
                .toLowerCase().split(SPLITTER.pattern());
```

Now rerunning the test, the result is the following:



```
[redacted]
country
joebiden
promise
[redacted]
lead
[redacted]
america
chosen
```

Therefore, this is working properly as the output is as expected and there are no errors/issues from the code.

Review

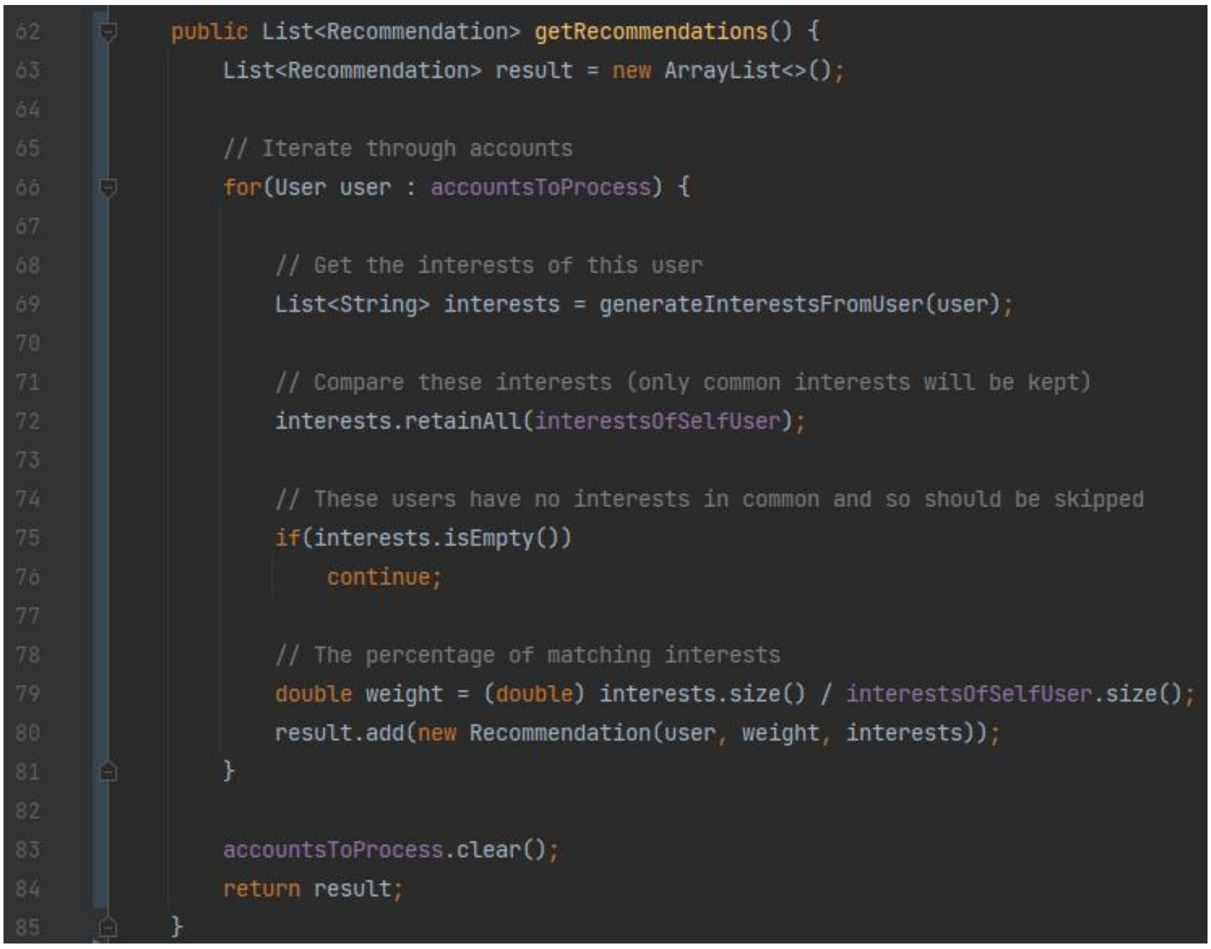
From this second iteration of the recommendation module, I will mark requirements that have been partially met in yellow/amber or fully completed in green. This was by far the most challenging iteration so far, as the algorithm to get Twitter accounts was very complex – especially when challenging myself to make it run asynchronously. I am very happy with the result, and I think it works well and will improve usability as the program should hopefully always be responsive. While many criteria remain the same, these will mostly be implemented during the GUI's implementation as they focus on the features of the display.

Requirement Number	Requirement	Justification	Success Criteria & Evidence	Provided Evidence
2. Recommendation module				
2.1	Rectangular boxes stacked vertically	Text is read left-to-right and therefore is more readable	Screenshot of correct layout	
2.2	Boxes expand upon being clicked	To give more information on a particular account, the box will expand once clicked to display this	Box must show more information when the user clicks it. Screenshot of expanded information view	
2.3	Boxes display information about recommendations	Information such as the similarities in interests, account name, etc will allow the user to decide whether to follow	Screenshots of information being displayed	
2.4 🟩	List of interests based on user	To give recommend-dations, a list of interests is needed about the account, which is gained via the Keyword Extraction Algorithm	The algorithm is given a text and will return a list of strings that are the important parts of the text. Screenshots of the input and output texts	The tweets of a user are retrieved and passed into the algorithm to return a list of interests/most frequent terms. There are screenshots + tests to support this
2.5	Follow button	Saves time for the user as they can quickly follow the account from the program	Once the button is pressed, the user will now be following the account on Twitter. Screenshots of before vs after button press	
2.6	Ignore button	Allows the user to block the account if they do not want to see it again	Once the account is blocked, they will never be recommended to the user again	

2.7	Interest search bar	Allows the user to search for a particular subject/topic they are interested in	Once searched, the user will receive recommend-dations based on that interest. Screenshots of relevant accounts being displayed	
🟩 2.8	Stop words	Stop words are words that will be excluded by the algorithm and so will save processing time as non-important words will not be processed	Algorithm does not output any stop words with pictures to demonstrate this working	The algorithm removes words that are listed in the stopwords text file. Screenshots were provided and tests carried out to prove this
🟩 2.9	Match languages	Only processes texts/tweets that are English (including US English) to match the stakeholder's language and reduce inaccurate recommendations	Algorithm should exclude processing texts from users that are not English speaking	The code validates that the languages must match for the accounts tweets to be processed. Screenshots provided show this

Iteration four – comparing and displaying recommendations

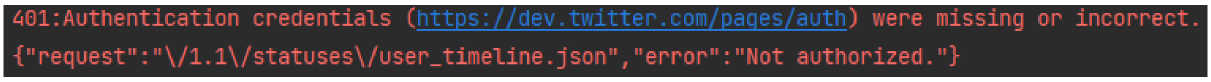
In this iteration, I will focus on narrowing down the recommendations to leave only ones that are relevant to the user. This will be done by comparing interests. Using this function, I could generate a list of recommendations:



```java
    public List<Recommendation> getRecommendations() {
        List<Recommendation> result = new ArrayList<>();

        // Iterate through accounts
        for(User user : accountsToProcess) {

            // Get the interests of this user
            List<String> interests = generateInterestsFromUser(user);

            // Compare these interests (only common interests will be kept)
            interests.retainAll(interestsOfSelfUser);

            // These users have no interests in common and so should be skipped
            if(interests.isEmpty())
                continue;

            // The percentage of matching interests
            double weight = (double) interests.size() / interestsOfSelfUser.size();
            result.add(new Recommendation(user, weight, interests));
        }

        accountsToProcess.clear();
        return result;
    }
```

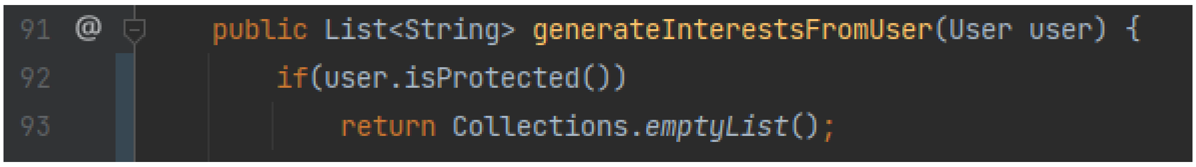
Recommendations are based on shared/common interests between the user and the account in question. This is found by only retaining the strings/interests that are present in the list of interests of the user (Removing all strings that are not present in the interests of the user list). Then, a weight is calculated based on the percentage of matching interests.

While testing, I kept encountering this error in console when fetching the tweets by an account:



```
401:Authentication credentials (https://dev.twitter.com/pages/auth) were missing or incorrect.
{"request":"\/1.1\/statuses\/user_timeline.json","error":"Not authorized."}
```

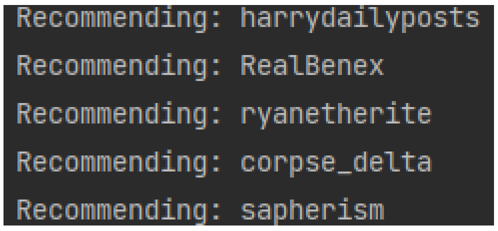
Upon further testing, this is due to the users account being private. Other people (unless they are following the account) cannot see the tweets of this user. As a result, my program cannot fetch the tweets, and this is what is causing the issue. To fix this, I will ignore any accounts that are private.



```java
@   public List<String> generateInterestsFromUser(User user) {
        if(user.isProtected())
            return Collections.emptyList();
```

After testing, this has fixed the issue as the error no longer appears in console.

After running the code, the algorithm returns a list of account names that it recommends you follow!



```
Recommending: harrydailyposts
Recommending: RealBenex
Recommending: ryanetherite
Recommending: corpse_delta
Recommending: sapherism
```

Review

As of now, this output is only to console, however as the GUI gets coded in the next iterations, this will no longer happen. While this iteration was short, it tied together a lot of the work from the previous two iterations to produce a successful output as desired. This iteration did not meet any of the success criteria, however it tied together previous work and the program would not produce an output without this iteration.

Requirement Number	**Requirement**	**Justification**	**Success Criteria & Evidence**	**Provided Evidence**
2. Recommendation module				
2.1	Rectangular boxes stacked vertically	Text is read left-to-right and therefore is more readable	Screenshot of correct layout	
2.2	Boxes expand upon being clicked	To give more information on a particular account, the box will expand once clicked to display this	Box must show more information when the user clicks it. Screenshot of expanded information view	
2.3	Boxes display information about recommendations	Information such as the similarities in interests, account name, etc will allow the user to decide whether to follow	Screenshots of information being displayed	
🟩 2.4	List of interests based on user	To give recommend-dations, a list of interests is needed about the account, which is gained via the Keyword Extraction Algorithm	The algorithm is given a text and will return a list of strings that are the important parts of the text. Screenshots of the input and output texts	The tweets of a user are retrieved and passed into the algorithm to return a list of interests/most frequent terms. There are screenshots + tests to support this
2.5	Follow button	Saves time for the user as they can quickly follow the	Once the button is pressed, the user will now be	

		account from the program	following the account on Twitter. Screenshots of before vs after button press	
2.6	Ignore button	Allows the user to block the account if they do not want to see it again	Once the account is blocked, they will never be recommended to the user again	
2.7	Interest search bar	Allows the user to search for a particular subject/topic they are interested in	Once searched, the user will receive recommend-dations based on that interest. Screenshots of relevant accounts being displayed	
🟩 2.8	Stop words	Stop words are words that will be excluded by the algorithm and so will save processing time as non-important words will not be processed	Algorithm does not output any stop words with pictures to demonstrate this working	The algorithm removes words that are listed in the stopwords text file. Screenshots were provided and tests carried out to prove this
🟩 2.9	Match languages	Only processes texts/tweets that are English (including US English) to match the stakeholder's language and reduce inaccurate recommendations	Algorithm should exclude processing texts from users that are not English speaking	The code validates that the languages must match for the accounts tweets to be processed. Screenshots provided show this

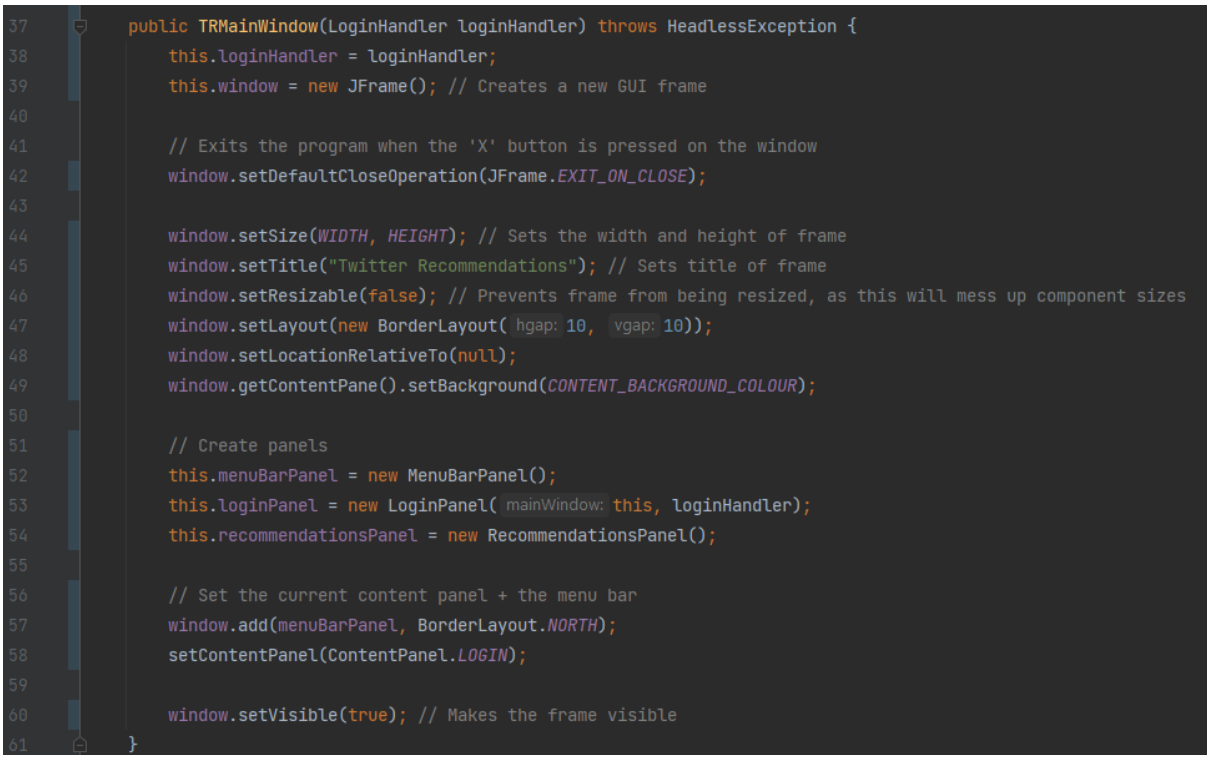
The Graphical User Interface

The next section to be developed is the GUI. This is important as it is the only way the user will interact with the program. To start with I will code the login screen, as this is the first thing a new user will be greeted with.

Iteration five – the login screen

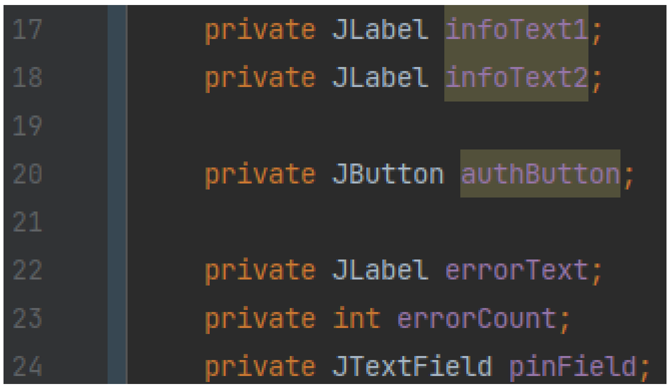
For my GUI, I have a main window object. All other components (login screen, recommendations screen, settings screen) are under separate panels. This means that the design of my GUI is modular in nature as the code for each panel is separated under individual classes/files.

Shown below is the main window file. This is the constructor method, which sets up the window, setting values such as the width, height, title, and the operation performed when pressing the 'X' at the top right. It then creates the individual panels. The menu par panel holds the users account information and button to visit settings and is always visible.

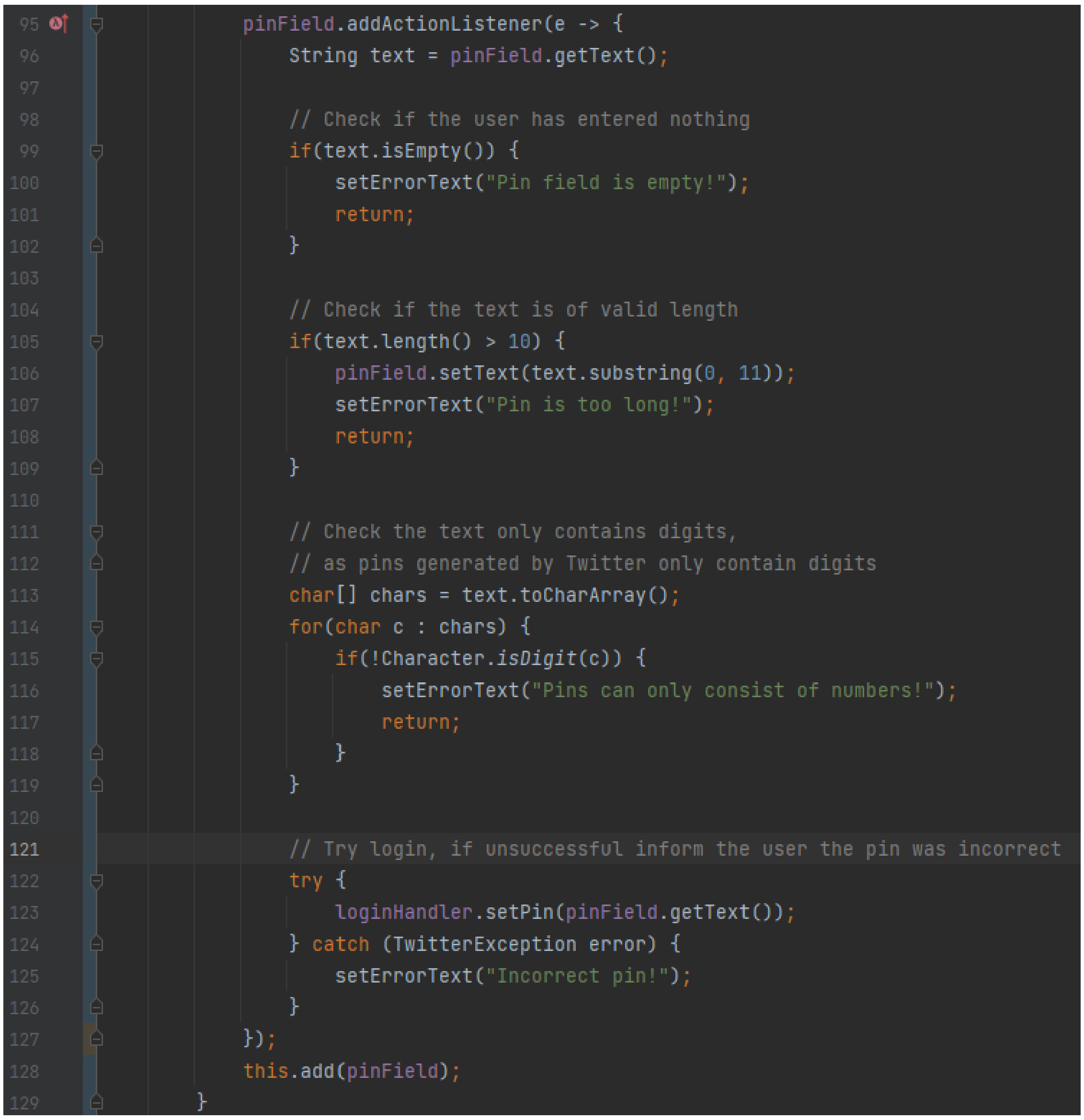


```
public TRMainWindow(LoginHandler loginHandler) throws HeadlessException {
    this.loginHandler = loginHandler;
    this.window = new JFrame(); // Creates a new GUI frame

    // Exits the program when the 'X' button is pressed on the window
    window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    window.setSize(WIDTH, HEIGHT); // Sets the width and height of frame
    window.setTitle("Twitter Recommendations"); // Sets title of frame
    window.setResizable(false); // Prevents frame from being resized, as this will mess up component sizes
    window.setLayout(new BorderLayout( hgap: 10,  vgap: 10));
    window.setLocationRelativeTo(null);
    window.getContentPane().setBackground(CONTENT_BACKGROUND_COLOUR);

    // Create panels
    this.menuBarPanel = new MenuBarPanel();
    this.loginPanel = new LoginPanel( mainWindow: this, loginHandler);
    this.recommendationsPanel = new RecommendationsPanel();

    // Set the current content panel + the menu bar
    window.add(menuBarPanel, BorderLayout.NORTH);
    setContentPanel(ContentPanel.LOGIN);

    window.setVisible(true); // Makes the frame visible
}
```

The first panel to be developed is the login screen, as this is the first thing the user will be greeted with when using my program. The login screen is comprised of these components. The info texts give instructions to the user on how to interact, the auth button opens Twitters authentication website in the users default browser and the pin field plus error variables allow the user to enter their pin as well as feedback any errors to the user.



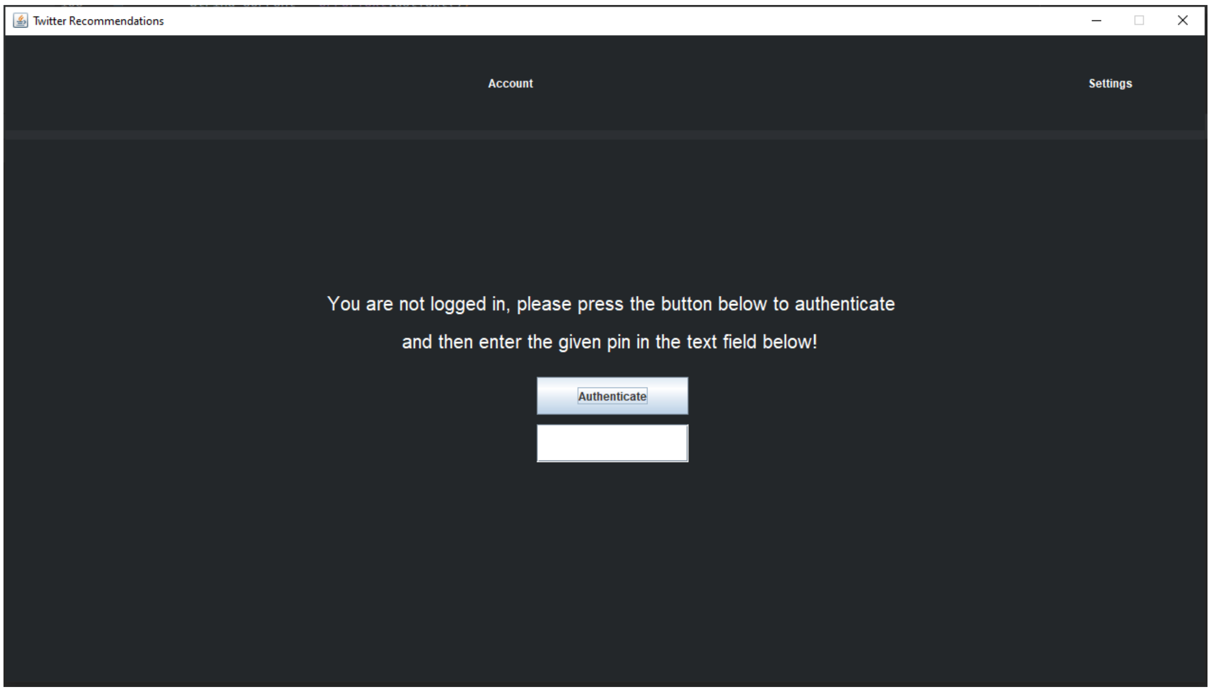
```
    private JLabel infoText1;
    private JLabel infoText2;

    private JButton authButton;

    private JLabel errorText;
    private int errorCount;
    private JTextField pinField;
```



```java
            pinField.addActionListener(e -> {
                String text = pinField.getText();

                // Check if the user has entered nothing
                if(text.isEmpty()) {
                    setErrorText("Pin field is empty!");
                    return;
                }

                // Check if the text is of valid length
                if(text.length() > 10) {
                    pinField.setText(text.substring(0, 11));
                    setErrorText("Pin is too long!");
                    return;
                }

                // Check the text only contains digits,
                // as pins generated by Twitter only contain digits
                char[] chars = text.toCharArray();
                for(char c : chars) {
                    if(!Character.isDigit(c)) {
                        setErrorText("Pins can only consist of numbers!");
                        return;
                    }
                }

                // Try login, if unsuccessful inform the user the pin was incorrect
                try {
                    loginHandler.setPin(pinField.getText());
                } catch (TwitterException error) {
                    setErrorText("Incorrect pin!");
                }
            });
            this.add(pinField);
        }
```

It was very important to validate the user's input here, to feedback information to the user as well as reducing the number of calls to the Twitter API. In the image above, it can be seen that I validate whether they have indeed entered any text before pressing enter, the text is of the right length, and whether the string is only digits. Finally, if the pin is incorrect but meets all of the above requirements, the user is informed that they may have entered the incorrect pin.

This produced the following result for the login screen:

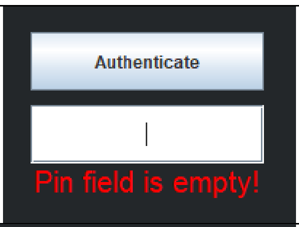
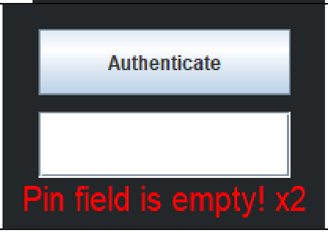


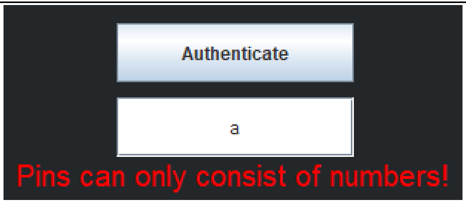
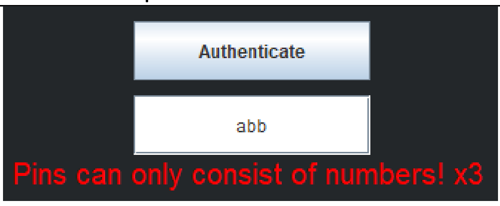
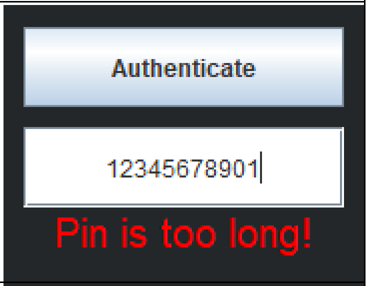
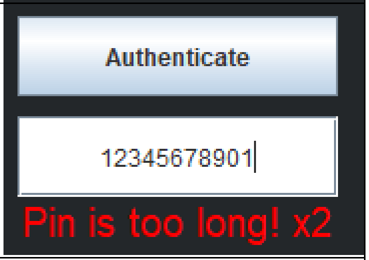
You are not logged in, please press the button below to authenticate

and then enter the given pin in the text field below!

When pressing the Authenticate button, Twitter's authentication webpage is opened in the user's default browser. Furthermore, the text entry field below allows the user to enter the pin. As can be seen, the colour scheme is dark mode, as described in the usability section of my design.

Before moving on, I will ensure this page works by testing its features.

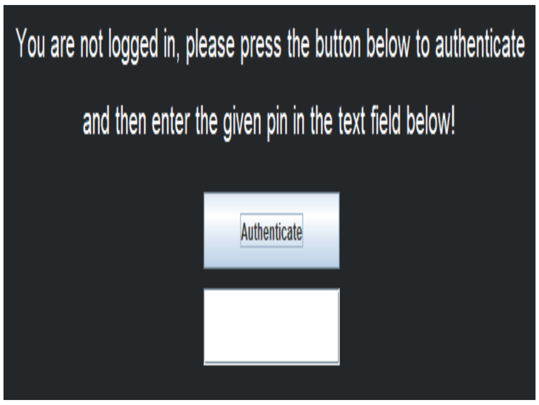
Test Case	Expected Result	Actual Result
Authentication button is pressed	Twitter's authentication webpage opens in default browser	Result as expected
Pin text field is submitted without any string	Red error message pops up informing user they need to enter a pin	Result as expected  Pin field is empty!
Pin text field is submitted without any string twice in a row	Red error message stays the same, but a count is shown to give feedback to the user that there input was acknowledged	Result as expected  Pin field is empty! x2

Pin text is submitted but contains an alphabetic character	Error message changes, and the count will disappear as it is a different error message.	 a Pins can only consist of numbers! Result as expected
Pin text is submitted but contains an alphabetic character (and the input has changed)	Error message stays the same and the count starts to give feedback to user	 abb Pins can only consist of numbers! x3 Result as expected
A pin text is submitted that is valid, but is too long	Error message shows	Result as expected  12345678901 Pin is too long!
A pin text is submitted that is valid, but is too long (Repeated)	Error message shows with count	Result as expected  12345678901 Pin is too long! x2
The correct pin is submitted	The screen changes from the login screen to the recommendation screen	Result as expected
Reopening the program once logged in	The program should automatically be opened to the recommendation screen	Result as expected

Next, to provide evidence for my success criteria, I will test various parts of the login screen and my program currently.

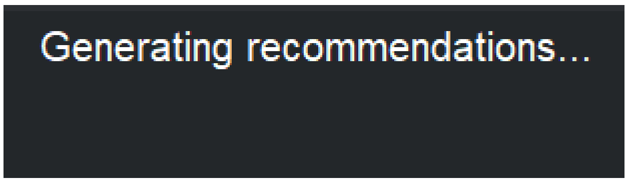
When loading the program for the first time (or if the user has not logged in before), they should be shown the login screen. Once they have authenticated, when they next load the program, it should not display the login screen. Instead, it should go straight to the recommendations screen.

Before logging in:



You are not logged in, please press the button below to authenticate and then enter the given pin in the text field below!

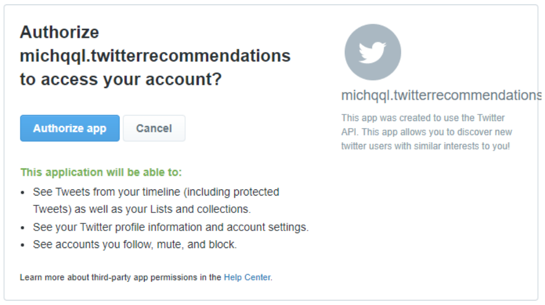
After logging in:



Generating recommendations...

As shown, once logged in the user is no longer greeted by the login screen.

Next, when the user presses the Authenticate button their default browser is opened to Twitter's authentication webpage. As shown below, this was opened in my default web browser (Chrome instead of Windows' default of Edge).



Authorize michqql.twitterrecommendations to access your account?

michqql.twitterrecommendations

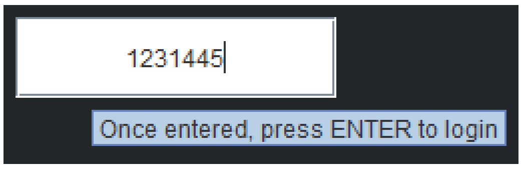
This app was created to use the Twitter API. This app allows you to discover new twitter users with similar interests to you!

This application will be able to:

- See Tweets from your timeline (including protected Tweets) as well as your Lists and collections.
- See your Twitter profile information and account settings.
- See accounts you follow, mute, and block.

Learn more about third-party app permissions in the Help Center.

Next, the pin text field. When hovering, a tool tip is given to the user to ensure they know how to use it properly.



1231445

Once entered, press ENTER to login

Review

From this first iteration of the GUI, I will mark requirements that have been partially met in yellow/amber or fully completed in green. As of now, the login page/module success criteria have been fully met. The only criteria that could be disputed as partially met is 1.3. This is because this is a first iteration, and this is something that can be altered later on to fully meet this criterion. The criteria are only partially met as the pin field was a text field instead of 10 individual buttons each representing their own number. However, in the design section I proposed two login screen layouts, one with buttons and one with a single text field. Therefore, it can be said that the design was met here. I think the GUI works well and it serves its purpose. In conclusion, the login success criteria have been met.

Requirement Number	Requirement	Justification	Success Criteria & Evidence	Provided Evidence
1. Login page/module				
🟩 1.1	Login UI is only shown to user if they are not logged in or it is the first-time using application	The user only needs to authenticate with Twitter once, and therefore this page only needs to be shown to the user once	Login UI is displayed first-time, but not again once they have authenticated. Screenshots of the menu taken to once loaded with and without login data	Test case that once the user has logged in, when reopening the program in the future it does show the login screen. As shown in the before vs after screenshots above
🟩 1.2	Button to open authentication in default browser	Using twitters website, the user must get an authentication code. Furthermore, using the default browser the user feels safe as this is their choice in browser	Clicking button opens users default browser to Twitter's authentication website. Screenshot of correct website	Screenshot shows the webpage open after clicking the Authenticate button
🟩 1.3	Number pin field (buttons)	Allows user to enter the authentication code provided by Twitter	Screenshot of buttons that user can press + their effect once pressed	Screenshots show the text field and demonstrates how the user interacts with them. Multiple screenshots show this text field
🟩 1.4	Logs in once valid authentication	The code will be used to log in with Twitters API and the rest of	Entering correct code changes UI screen to main recommendations.	Testing shows that when the correct pin is entered, the user is logged in properly and

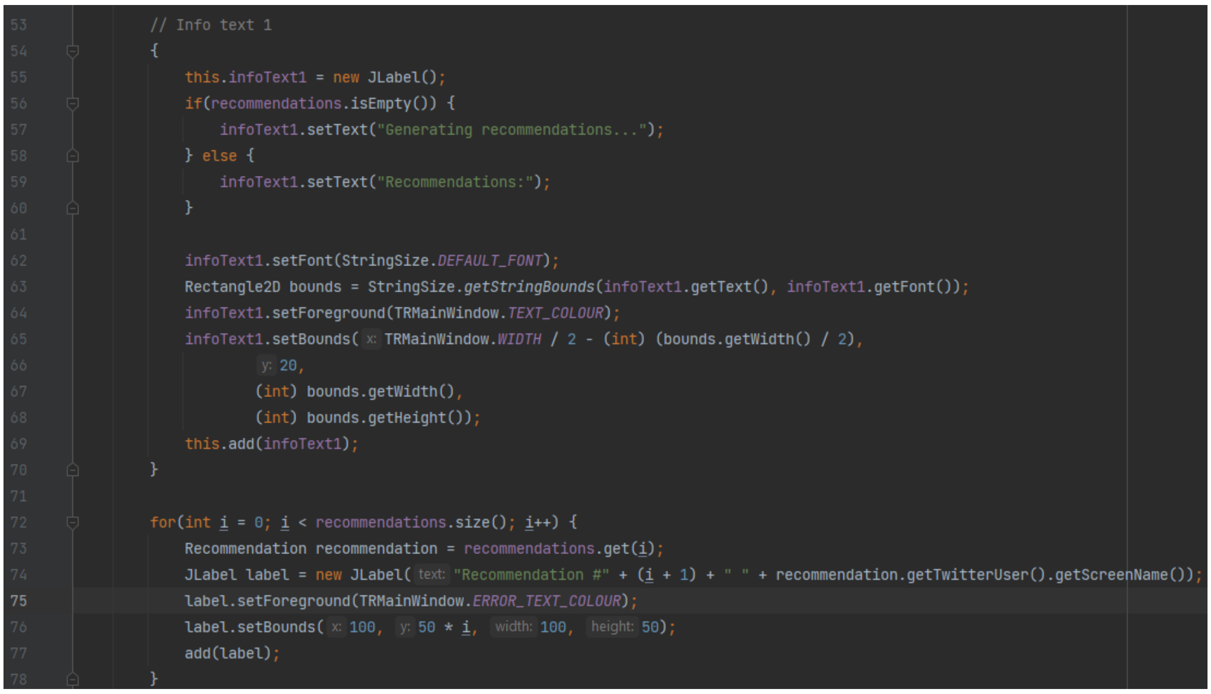
🟩	code is provided	the application can then be used. The user will be taken to the main screen	Screenshot of before + after entering correct code	the recommendations screen is then displayed to them. Screenshots above show the before vs after logging in
🟩 1.5	Authentication data is saved to file	This is required so that the user doesn’t need to authenticate each time the load the application	After the user has authenticated for the first time, the data regarding their account is stored to a file on their computer. Screenshot of stored data	Data relating to the user, the token and secret are all saved to “tokens.json”. As shown in screenshot in iteration one

Iteration six – the recommendation screen

The next GUI page to be programmed is the recommendation screen. This will be the screen that the user interacts with the most and will be shown after logging in.

Using the interest handler that I coded in an earlier iteration, I could now get recommendations and display them to the screen.

I first created this code



```java
        // Info text 1
        {
            this.infoText1 = new JLabel();
            if(recommendations.isEmpty()) {
                infoText1.setText("Generating recommendations...");
            } else {
                infoText1.setText("Recommendations:");
            }

            infoText1.setFont(StringSize.DEFAULT_FONT);
            Rectangle2D bounds = StringSize.getStringBounds(infoText1.getText(), infoText1.getFont());
            infoText1.setForeground(TRMainWindow.TEXT_COLOUR);
            infoText1.setBounds( x: TRMainWindow.WIDTH / 2 - (int) (bounds.getWidth() / 2),
                     y: 20,
                    (int) bounds.getWidth(),
                    (int) bounds.getHeight());
            this.add(infoText1);
        }

        for(int i = 0; i < recommendations.size(); i++) {
            Recommendation recommendation = recommendations.get(i);
            JLabel label = new JLabel( text: "Recommendation #" + (i + 1) + " " + recommendation.getTwitterUser().getScreenName());
            label.setForeground(TRMainWindow.ERROR_TEXT_COLOUR);
            label.setBounds( x: 100,  y: 50 * i,  width: 100,  height: 50);
            add(label);
        }
```

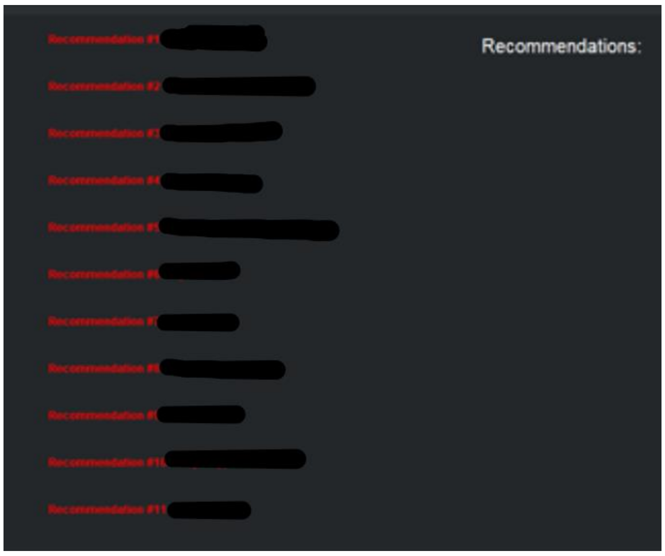
The info text tells the user of the current status of the program. If recommendations are being generated in the async thread, then the list will be empty, and it will inform the user that they are being generated currently. Otherwise, the recommendations will be displayed below using a loop.

Currently, this code is just to test. However, an issue I am having is that the screen does not update unless I minimise and open the program again, causing the GUI to update. This is a problem as the recommendations are generated after a certain delay, and it calls this method to redraw the components, however it does not update the screen.

After reading through some of the class's documentation, I found the method 'repaint'. This tells the component to redraw all children components. When calling this on the Java Panel component that holds all of the text, it successfully displays the new content. Therefore, this issue has been fixed.

This text is currently temporary and will be substituted out for the actual design later on in this iteration.

```
super.repaint();
```



[redacted]

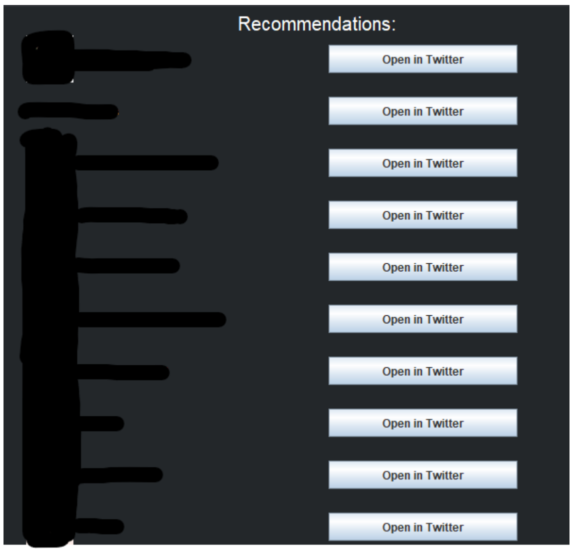
Next, I coded the final layout for this screen. After implementing the users name and profile pictures, I went to add a button to follow the user from my program, as specified in requirement 2.5 from my analysis section. However, I kept getting an error from Twitter when trying to do this, saying my program was not authenticated to do this action. Unfortunately, I could not solve this problem. I did not have permission from Twitter to make the user's account follow other accounts on Twitter.

```
401:Authentication credentials (https://dev.twitter.com/pages/auth) were missing or incorrect.
```

Therefore, my solution for this is shown below. Instead, I had to have a button to open the account in Twitter. From there, the user has the option to follow them anyway, creating a work around for this problem.

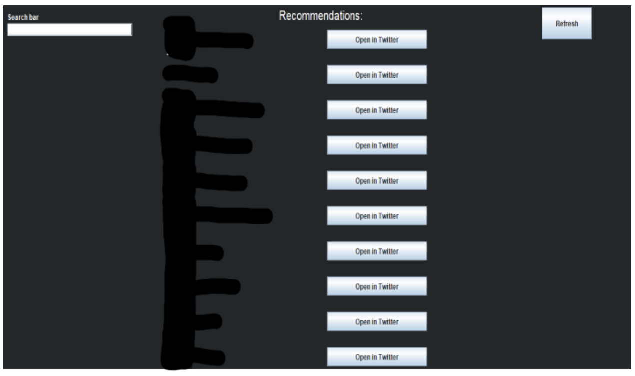
After a little coding and tweaking around with position values, I finished with this layout for this screen:

On the left, displays the account. If users have a profile picture, it will be displayed to the left followed by their display name. To the right, a button allows the user to open the account in Twitter so that they can view tweets, follow etc. The code for this is shown on the next page.



[redacted]

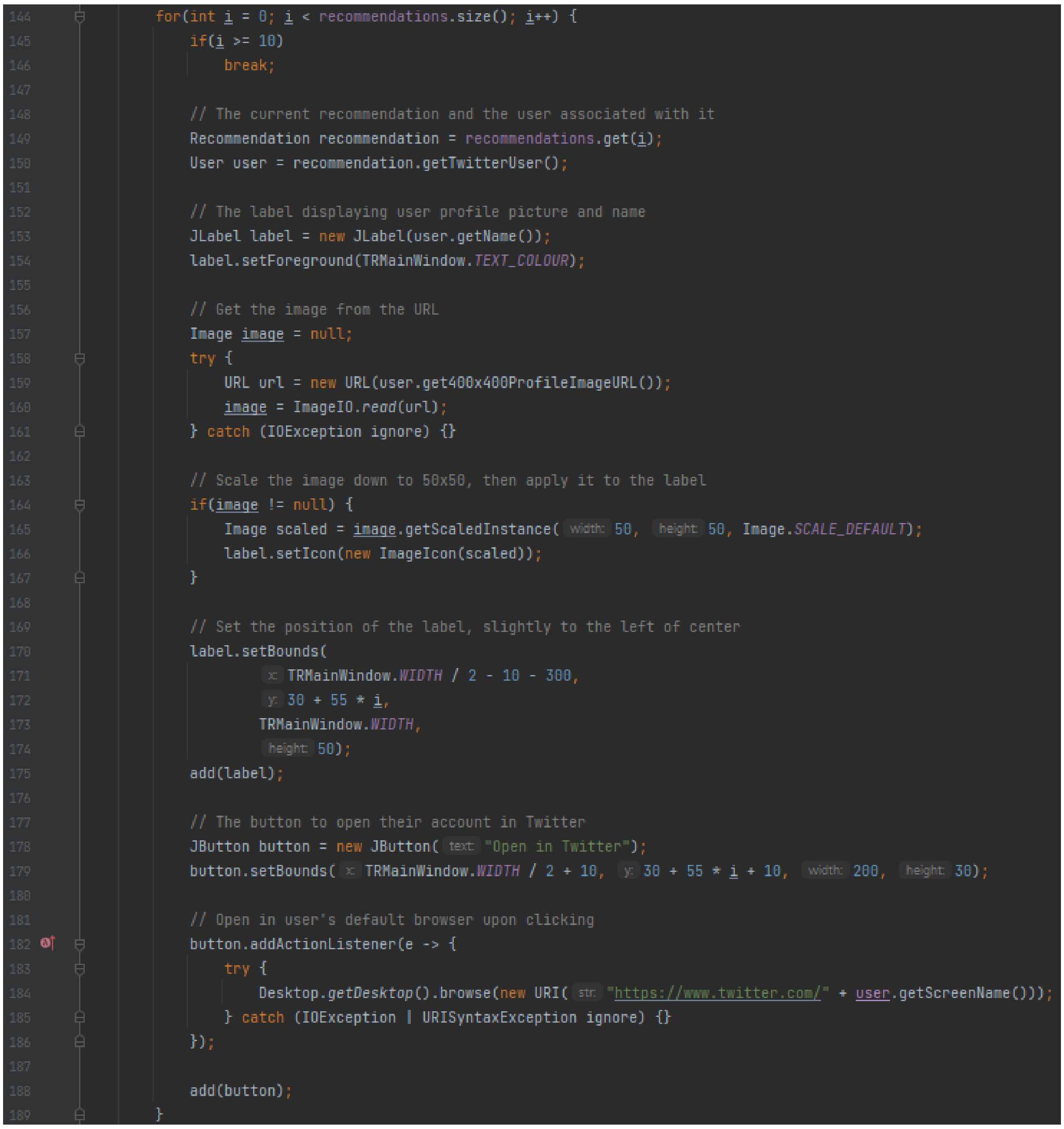
Next, I implemented a refresh button to meet requirement 2.10 from my analysis section. This generates a new set of recommendations. At the same time, I coded a search bar to meet requirement 2.7 from my analysis section. These are shown below.



[redacted]

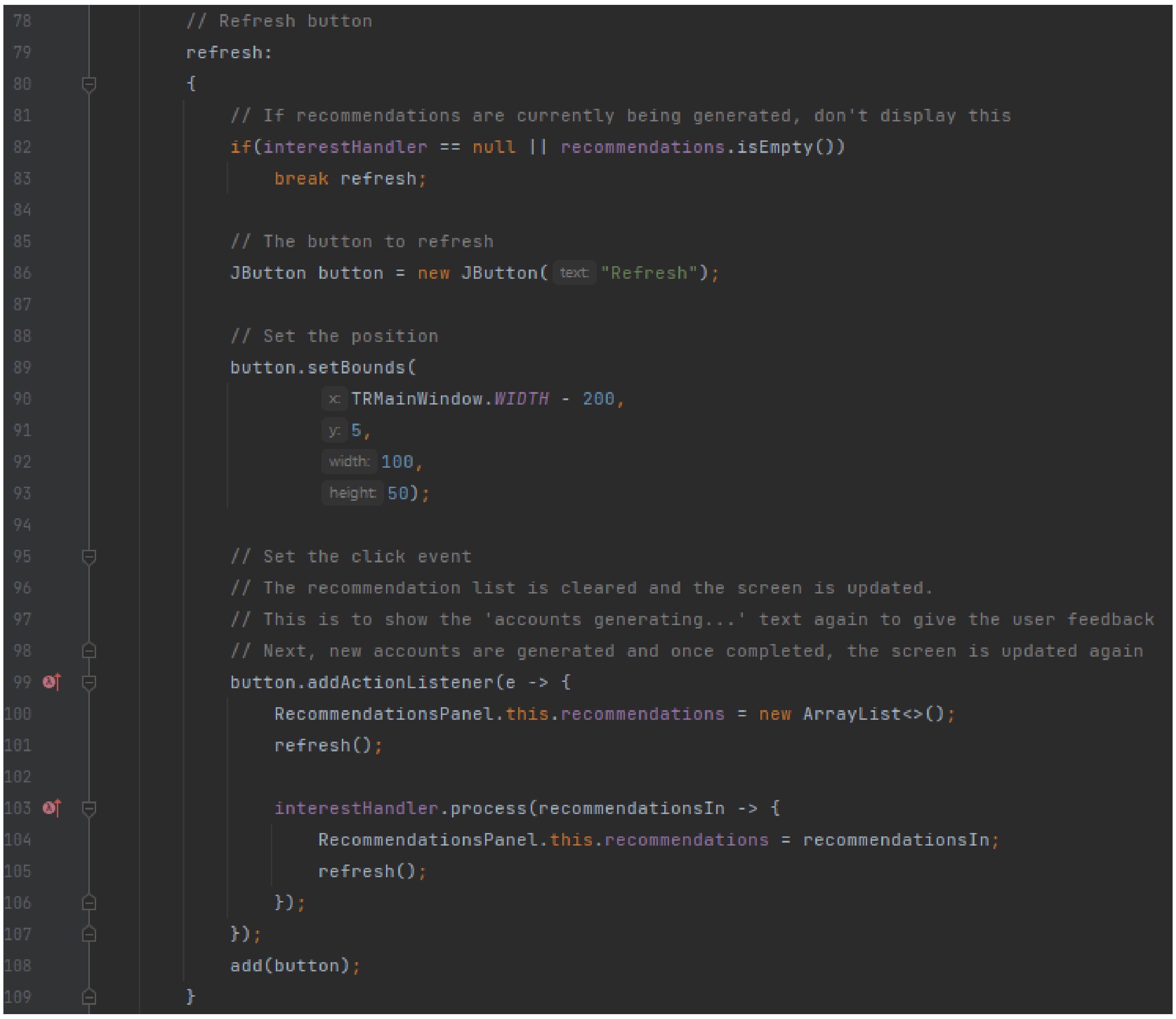
On the left, the search bar can be seen whereas on the right the refresh button is shown.

Code to display each recommendation, including information about the account and follow button.



```java
for(int i = 0; i < recommendations.size(); i++) {
    if(i >= 10)
        break;

    // The current recommendation and the user associated with it
    Recommendation recommendation = recommendations.get(i);
    User user = recommendation.getTwitterUser();

    // The label displaying user profile picture and name
    JLabel label = new JLabel(user.getName());
    label.setForeground(TRMainWindow.TEXT_COLOUR);

    // Get the image from the URL
    Image image = null;
    try {
        URL url = new URL(user.get400x400ProfileImageURL());
        image = ImageIO.read(url);
    } catch (IOException ignore) {}

    // Scale the image down to 50x50, then apply it to the label
    if(image != null) {
        Image scaled = image.getScaledInstance( width: 50,  height: 50, Image.SCALE_DEFAULT);
        label.setIcon(new ImageIcon(scaled));
    }

    // Set the position of the label, slightly to the left of center
    label.setBounds(
            x: TRMainWindow.WIDTH / 2 - 10 - 300,
            y: 30 + 55 * i,
            TRMainWindow.WIDTH,
            height: 50);
    add(label);

    // The button to open their account in Twitter
    JButton button = new JButton( text: "Open in Twitter");
    button.setBounds( x: TRMainWindow.WIDTH / 2 + 10,  y: 30 + 55 * i + 10,  width: 200,  height: 30);

    // Open in user's default browser upon clicking
    button.addActionListener(e -> {
        try {
            Desktop.getDesktop().browse(new URI( str: "https://www.twitter.com/" + user.getScreenName()));
        } catch (IOException | URISyntaxException ignore) {}
    });

    add(button);
}
```

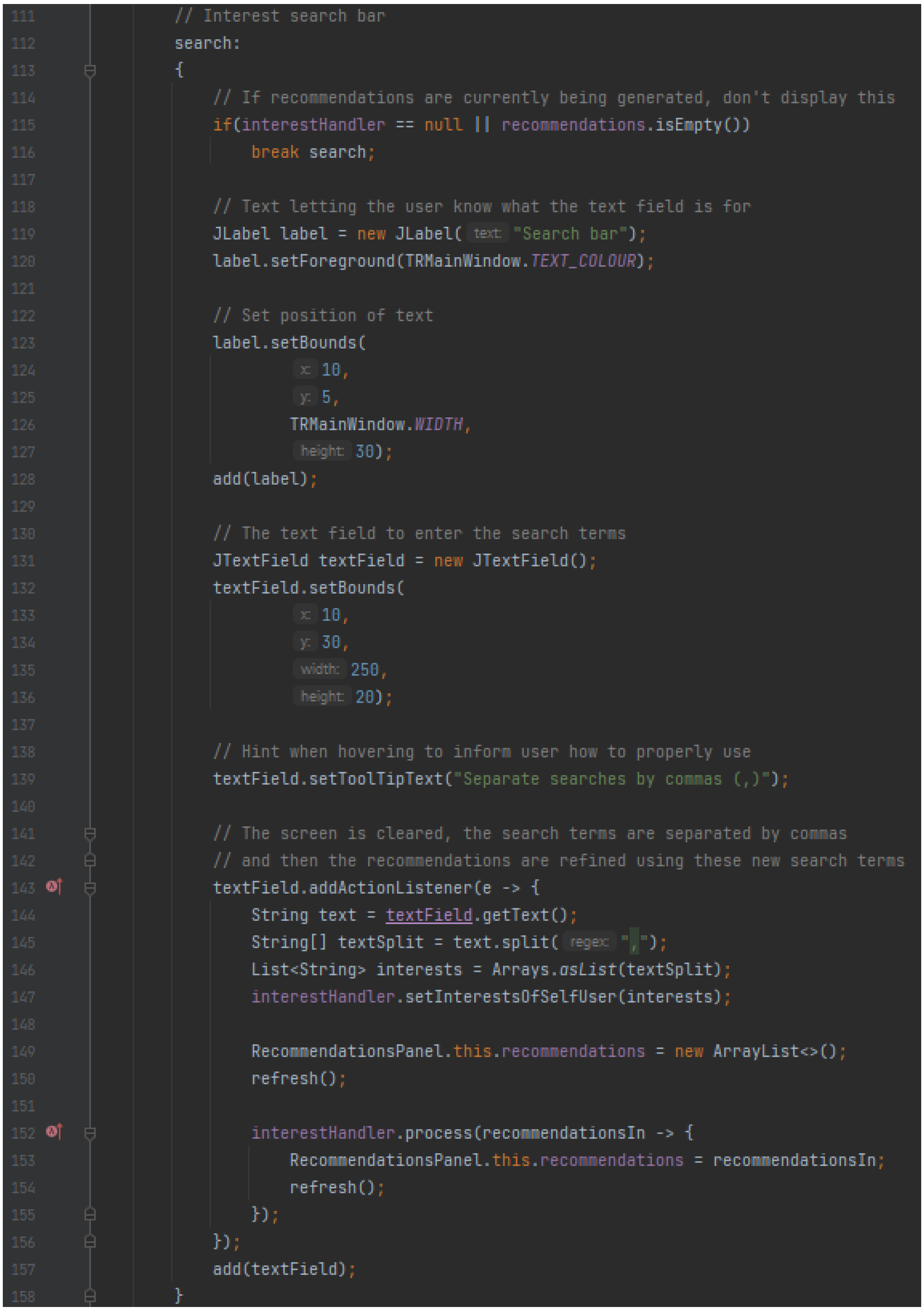
Below shows the code for the refresh button:



```java
        // Refresh button
        refresh:
        {
            // If recommendations are currently being generated, don't display this
            if(interestHandler == null || recommendations.isEmpty())
                break refresh;

            // The button to refresh
            JButton button = new JButton( text: "Refresh");

            // Set the position
            button.setBounds(
                    x: TRMainWindow.WIDTH - 200,
                    y: 5,
                    width: 100,
                    height: 50);

            // Set the click event
            // The recommendation list is cleared and the screen is updated.
            // This is to show the 'accounts generating...' text again to give the user feedback
            // Next, new accounts are generated and once completed, the screen is updated again
            button.addActionListener(e -> {
                RecommendationsPanel.this.recommendations = new ArrayList<>();
                refresh();

                interestHandler.process(recommendationsIn -> {
                    RecommendationsPanel.this.recommendations = recommendationsIn;
                    refresh();
                });
            });
            add(button);
        }
```

Once the refresh button has been clicked, the whole panel is updated to remove all the content and only show the ‘Generating Recommendations…’ text again. This is to provide the user with feedback that their input has done something.

Next, the code for the search bar:

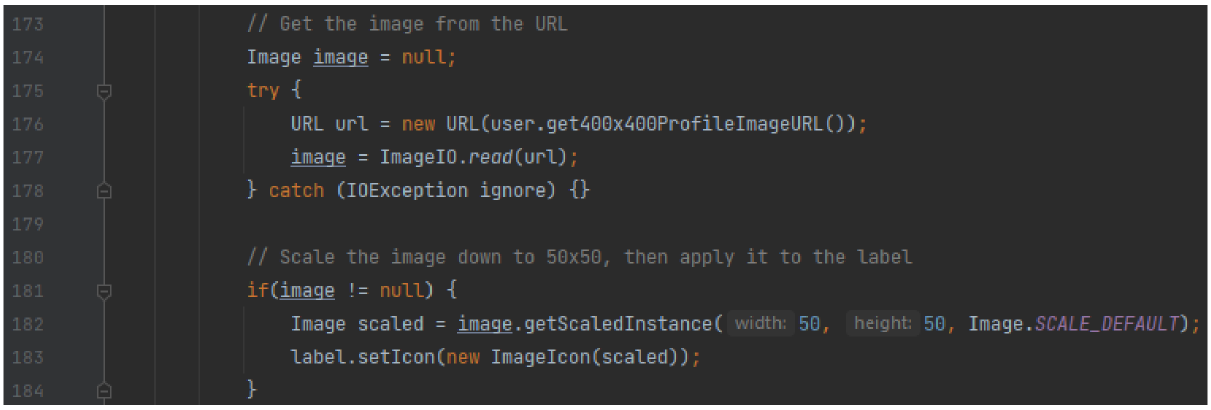


```
// Interest search bar
search:
{
    // If recommendations are currently being generated, don't display this
    if(interestHandler == null || recommendations.isEmpty())
        break search;

    // Text letting the user know what the text field is for
    JLabel label = new JLabel( text: "Search bar");
    label.setForeground(TRMainWindow.TEXT_COLOUR);

    // Set position of text
    label.setBounds(
            x: 10,
            y: 5,
            TRMainWindow.WIDTH,
            height: 30);
    add(label);

    // The text field to enter the search terms
    JTextField textField = new JTextField();
    textField.setBounds(
            x: 10,
            y: 30,
            width: 250,
            height: 20);

    // Hint when hovering to inform user how to properly use
    textField.setToolTipText("Separate searches by commas (,)");

    // The screen is cleared, the search terms are separated by commas
    // and then the recommendations are refined using these new search terms
    textField.addActionListener(e -> {
        String text = textField.getText();
        String[] textSplit = text.split( regex: ",");
        List<String> interests = Arrays.asList(textSplit);
        interestHandler.setInterestsOfSelfUser(interests);

        RecommendationsPanel.this.recommendations = new ArrayList<>();
        refresh();

        interestHandler.process(recommendationsIn -> {
            RecommendationsPanel.this.recommendations = recommendationsIn;
            refresh();
        });
    });
    add(textField);
}
```

Again, like the refresh button, when searching the screen is cleared to provide the user with feedback that their input has done something.

Validation

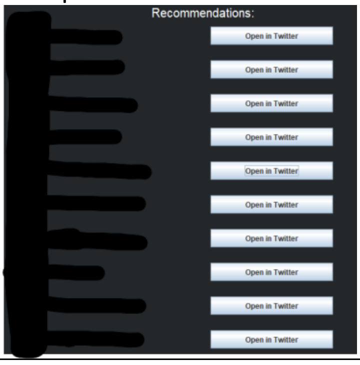
When displaying the profile pictures of accounts on the screen, if the account does not have one then this will be caught. Otherwise, an error would occur which could potentially crash the program.

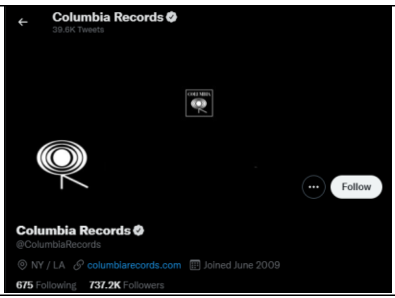
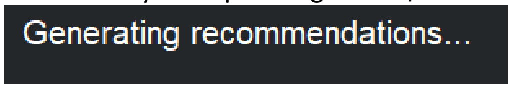
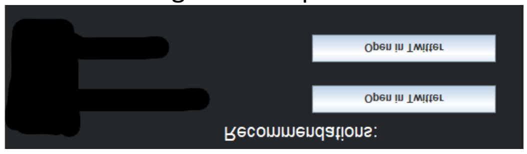
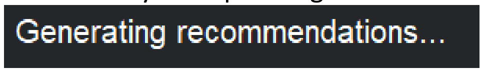
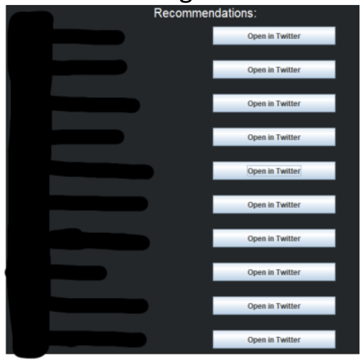


```java
// Get the image from the URL
Image image = null;
try {
    URL url = new URL(user.get400x400ProfileImageURL());
    image = ImageIO.read(url);
} catch (IOException ignore) {}

// Scale the image down to 50x50, then apply it to the label
if(image != null) {
    Image scaled = image.getScaledInstance( width: 50,  height: 50, Image.SCALE_DEFAULT);
    label.setIcon(new ImageIcon(scaled));
}
```

As can be seen in the screenshot above, if an error occurs when getting the image then the image object will still be null. Therefore, it will not attempt to display an image that doesn't exist.

Testing

Test Case	Expected Result	Actual Result
The recommendation screen is shown (either after successfully authenticating or loading the program after having previously authenticated)	User is greeted with text informing them that recommendations are being generated, then the recommendations are shown.	After a few seconds of waiting while the recommendations are generated, the user is shown this screen displaying all recommendations. Therefore, the result is as expected.  Recommendations: Open in Twitter Open in Twitter Open in Twitter Open in Twitter Open in Twitter Open in Twitter Open in Twitter Open in Twitter Open in Twitter Open in Twitter [redacted]
The user presses the 'Open in Twitter' button	In the user's default browser, the Twitter web page of the account they clicked on is opened.	As expected, the webpage opened in my default browser to the correct Twitter account. In this case, I was recommended Columbia Records, and therefore when clicking on that button, it opened in my browser.

		 Columbia Records 39.6K Tweets Follow Columbia Records @ColumbiaRecords NY / LA columbiarecords.com Joined June 2009 675 Following 737.2K Followers
The user searches for terms in the search bar	Once pressing enter, the screen switches to inform the user that new recommendations are being generated. Then, it displays the new recommendations.	Immediately after pressing search/ENTER: Generating recommendations... After searching for the input: [redacted] The search term was quite specific, as I only searched for 'music' and therefore only two accounts were found with that one specific term. It would work better with larger sets of terms, as that is how it is designed to work. The result is as expected.
The user presses the refresh button	Once pressing, the screen switches to inform the user that new recommendations are being generated. Then, it displays the new recommendations.	Immediately after pressing button: Generating recommendations... After refreshing: Recommendations: [redacted] Open in Twitter Open in Twitter Open in Twitter Open in Twitter Open in Twitter Open in Twitter Open in Twitter Open in Twitter Open in Twitter Open in Twitter Result is as expected, as a new list of recommendations are shown to the user.

Review

At the end of this iteration, the recommendation screen works great, and it recommends you the users. Overall, this iteration was a success, even though some problems were encountered such as not being able to follow the user directly through my program, however, a work around was created to open the account's Twitter page directly in the user's browser which allows them to follow.

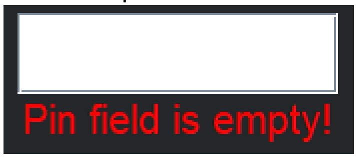
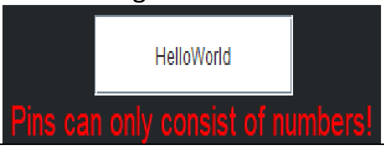
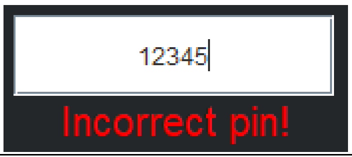
Requirement Number	Requirement	Justification	Success Criteria & Evidence	Provided Evidence
2. Recommendation module				
2.1	Rectangular boxes stacked vertically	Text is read left-to-right and therefore is more readable	Screenshot of correct layout	Screenshots show that recommendations are stacked vertically on top of each other
2.2	Boxes expand upon being clicked	To give more information on a particular account, the box will expand once clicked to display this	Box must show more information when the user clicks it. Screenshot of expanded information view	
2.3	Boxes display information about recommendations	Information such as the similarities in interests, account name, etc will allow the user to decide whether to follow	Screenshots of information being displayed	
2.4	List of interests based on user	To give recommend-dations, a list of interests is needed about the account, which is gained via the Keyword Extraction Algorithm	The algorithm is given a text and will return a list of strings that are the important parts of the text. Screenshots of the input and output texts	The tweets of a user are retrieved and passed into the algorithm to return a list of interests/most frequent terms. There are screenshots + tests to support this. This is proven in iteration four
2.5	Follow button	Saves time for the user as they can quickly follow the account from the program	Once the button is pressed, the user will now be following the account on Twitter. Screenshots of before vs after button press	Screenshots show that a button to open the user's account in Twitter on the browser is provided, however due to limitations in the Twitter API it was not possible to

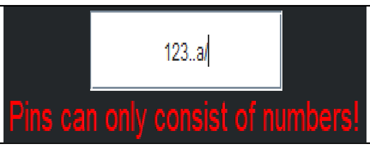
🟧				directly follow from my program
2.6	Ignore button	Allows the user to block the account if they do not want to see it again	Once the account is blocked, they will never be recommended to the user again	
🟩 2.7	Interest search bar	Allows the user to search for a particular subject/topic they are interested in	Once searched, the user will receive recommend-dations based on that interest. Screenshots of relevant accounts being displayed	Screenshots show the search bar and its functionality. And test cases prove that it works as intended.
🟩 2.8	Stop words	Stop words are words that will be excluded by the algorithm and so will save processing time as non-important words will not be processed	Algorithm does not output any stop words with pictures to demonstrate this working	The algorithm removes words that are listed in the stopwords text file. Screenshots were provided and tests carried out to prove this. This is proven in iteration four
🟩 2.9	Match languages	Only processes texts/tweets that are English (including US English) to match the stakeholder's language and reduce inaccurate recommendations	Algorithm should exclude processing texts from users that are not English speaking	The code validates that the languages must match for the accounts tweets to be processed. Screenshots provided show this. This is proven in iteration four

Evaluation

Post development testing

Testing possible inputs for the pin text field on the login screen.
All of these test cases have been covered and annotated in iteration five

Testing	Data	Expected outcome	Actual result
Token is empty	Token = “”	User told to authenticate	Error message is shown to user informing them they must enter the pin.  Pin field is empty! As can be seen in the image, below the pin text input box, the error message displays to the user to inform them of the issue. This is the same for all outcomes.
Token is valid	Token = “Gs5Xa”	Success	The login screen is never shown to the user, as they have already authenticated. The recommendation screen is shown to the user.
Auth code is valid	Code = “123456”	Success	The recommendation screen is shown to the user and their authentication data is saved.
Auth code is text	Code = “Apples”	Error thrown	Error message is shown to user informing them the pin only contains digits.  HelloWorld Pins can only consist of numbers!
Auth code is wrong	Code = “000000”	Error thrown	Error message is shown to user informing them the pin is incorrect.  12345 Incorrect pin!
Auth code is invalid	Code = “.../*&”	Error thrown	Error message is shown to user informing them the pin only contains digits.

			 123..a| Pins can only consist of numbers!

In these test cases, the functionality is very good as the program does not crash, but instead informs the user of the problem. In terms of robustness, it is also good as all cases inform the user giving the reason why the problem has occurred.

Testing possible values for the keyword extraction algorithm.

Testing	**Data**	**Expected outcome**	**Actual result**
Text is empty	Text = “”	Nothing	As expected, there is no result
Text is numbers	Text = 3	Error thrown	The number is removed from the text, and therefore does not throw and error. This is a better outcome that what was expected
Text is valid	Text = tweet	Term frequency determined	As expected, this works as normal as this is a term that can be processed
Text is null	Text = null	Error thrown	It is not possible for the text of a tweet to be null, however, if a null value was passed an error occurred.

The functionality of the keyword extraction algorithm is good as it automatically filters out any non-terms. This prevents errors and produces a good result.

Testing possible inputs for the interest search bar on the recommendation screen.

Testing	**Data**	**Expected outcome**	**Actual result**
String is passed	Data = “Football”	Success	The input is processed as expected as it is a term
Integer is passed	Data = 3	Error thrown	The term is removed from the search list, but the other terms are still used. No error is thrown.
Float is passed	Data = 3.1	Error thrown	The term is removed from the search list, but the other terms are still used. No error is thrown.
Non-alphabetic character is input	Data = /.,” %&	Error thrown	The term is removed from the search list, but the other terms are still used. No error is thrown.

Next, the test scenarios that were laid out at the end of the design section will be tested by the stakeholders. The thoughts and impressions of each stakeholder for their scenarios are detailed below the scenario.

Test Scenario 1: New user (Emma)

- Open the program
- Authenticate with Twitter
- Enter correct code
- Interact with recommendations
- Open in browser
- Close the program

Thoughts:

- The login screen was intuitive as it provided information and feedback of how-to login
- Recommendations could have provided more information about the user, but it is okay that a button is provided to view their Twitter, as this also gives information about the account

Test Scenario 2: Average user (Chirag, Sam)

- Open the program
- Authenticate
- Enter code
- Interact with recommendations
- Block account
- View more information about an account
- Search for a specific interest
- Open in browser
- Close the program

Thoughts:

- The login screen was easy to use and understand
- I was not able to block an account as this was not a feature, however, I learned that if I blocked them on Twitter, they would no longer be recommended for me
- I was able to view more information about an account using the button to open in Twitter
- Searching for interests was easy, and the hint was useful to give me more knowledge into how to use the program

Test Scenario 3: Advanced user (Fin)

- Open the program
- Authenticate
- Search for specific interest regarding trending topics
- Open in browser
- View metrics
- Expand information
- Search for different interest

- Open settings GUI
- Unblock user
- Ignore certain interest
- Return to recommendations
- Search for ignored interest
- Close the program

Thoughts:

- I liked the login screen, as the messages were very useful, and it gave feedback if what I entered was incorrect
- Recommendation screen could have been improved, as it did not provide me with much more than the basic account information
- I was not able to view metrics as this was not a feature
- I was able to view more information by opening in Twitter
- Searching was easy to do

Feedback

From this feedback from my stakeholders, I understand that the recommendation page could have been better implemented.

Cross reference with Success Criteria

This is my success criteria copied from my analysis section. Any fully met criteria will be coloured green, partially met in amber, and not met will be in red.

Requirement Number	Requirement	Evidence (link to page)	Success Criteria	Evaluate and explain
1. Login page/module				
🟩 1.1	Login UI is only shown to user if they are not logged in or it is the first-time using application	Iteration five	Login UI is displayed first-time, but not again once they have authenticated	This was a success as the post development testing shows that the Stakeholders thought the login screen was easy to use. This has been met as the user was not greeted with the login screen once they authenticated once.
🟩 1.2	Button to open authentication in default browser		Clicking button opens users default browser to Twitter's	This was fully met as the button opened Twitter's

🟩			authentication website	authentication website on the user's default browser.
🟧 1.3	Number pin field (buttons)		Screenshot of buttons that user can press + their effect once pressed	This was partially met as the design was not perfect, but the test evidence shows that the stakeholders found it easy to use.
🟩 1.4	Logs in once valid authentication code is provided		Entering correct code changes UI screen to main recommendations	This was fully met, as once the correct pin had been entered, the user was taken to the recommendation screen.
🟩 1.5	Authentication data is saved to file		After the user has authenticated for the first time, the data regarding their account is stored to a file on their computer	This was fully met as the authentication data is saved to a json file once the user authenticated.
2. Recommendation module				
🟩 2.1	Rectangular boxes stacked vertically	Iteration six	Screenshot of correct layout	This was fully met as the recommendations were vertically laid out.
🟥 2.2	Boxes expand upon being clicked		Box must show more information when the user clicks it	This was not met, and stakeholders felt that more information would have been nice, as the currently displayed information was basic.
🟩 2.3	Boxes display information about recommendations		Screenshots of information being displayed	This was fully met as there was information about the accounts being recommended, such as profile picture and name.
🟩 2.4	List of interests based on user	Iteration three	The algorithm is given a text and will return a list of strings that are the important parts of the text	This was fully met as the interests were generated based on the user's tweets.
🟧 2.5	Follow button	Iteration six	Once the button is pressed, the user will	This was partially met, as mentioned in iteration six, due to a

🟨			now be following the account on Twitter	technical issue with Twitter's API, this was not possible to fully meet. Therefore, a work around had to be implemented to open the account in the Twitter website. However, this still only partially met the success criteria.
🟥 2.6	Ignore button		Once the account is blocked, they will never be recommended to the user again	This was not met as the feature was not implemented, however from stakeholder feedback, no one mentioned this as an issue.
🟩 2.7	Interest search bar		Once searched, the user will receive recommend-dations based on that interest	This was fully met as the search bar was provided on the recommendation page and was easy to use and worked well. Stakeholders liked this feature and said it was easy to use.
🟩 2.8	Stop words	Iteration two	Algorithm does not output any stop words with pictures to demonstrate this working	This was fully met as a long data set of stopwords were used to filter out basic words that would not give information of a Tweet.
🟩 2.9	Match languages	Iteration three	Algorithm should exclude processing texts from users that are not English speaking	This was met, as only accounts that spoke the same language as the user were recommended.
🟩 2.10	Refresh button	Iteration six	Upon pressing the refresh button, the user should have feedback that their input did something while new recommendations are generated	This was fully met as the refresh button was provided on the recommendation page. Stakeholders liked this as it meant they could generate

🟩				more recommendations.

Addressing further development

🟩 1.1	🟩 1.2	🟨 1.3	🟩 1.4	🟩 1.5	🟩 2.1	🟥 2.2	🟩 2.3	🟩 2.4	🟨 2.5	🟥 2.6	🟩 2.7	🟩 2.8	🟩 2.9	🟩 2.10

Some features were only partially or not met. This is due to a few reasons. However, on the whole 73% of the features mentioned in the success criteria were met.

For requirement 1.3, in the future I would program all 10 buttons for each number 0-9 to allow the user to enter the pin in that style. This could have been achieved if I had more time, however due to time constraints, I had to settle with the other mock design from my design section.

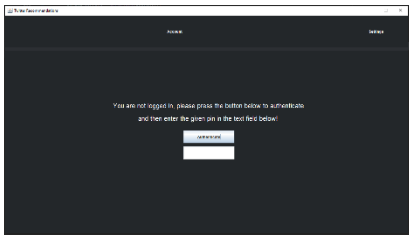
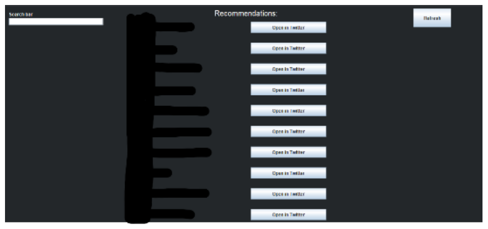
For requirement 2.2, a combination between lack of experience working with Java Swing GUI coding and time constraints meant that this was not met. In the future, I would like to have taken more time to fully educate myself into GUI coding in Java in order to have to knowledge to code something like this. Perhaps at the time of writing my analysis, I overestimated my abilities and designed a GUI that was far to big for the time scale and scope of this project.

For requirement 2.5, this was partially met but was limited due to a technical limitation in Twitter's API. My program did not have the permission from Twitter to make follow requests on behalf of another user. Therefore, it was not possible for me to implement this feature fully. However, a work around was to provide a button to open the account on Twitter's website in the user's default browser. From there, the option to follow the account is presented to the user. In the future, to fix this, I would have contacted Twitter support to try and resolve this permission issue. However, due to time constraints, I did not have the time to go through Twitter's support (as previous interactions with Twitter support took over a month for a response). In further development, if I was granted this permission by Twitter, I would simply have another button beside each account that when pressed follows the account directly from my program.

For requirement 2.6, this feature was not met due to time constraints. In further development, this could be addressed by saving a list of account ID's that the user has ignored, and then when accounts are generated these ignored accounts are removed. Then, beside each user a button to ignore them would appear, once clicked the account would be added to the ignore list. The list would be saved to a file when the user exits the program and read from when reopened.

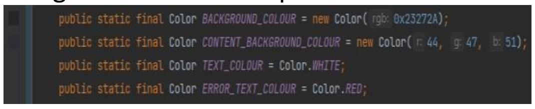
Usability features

This table is copied from my design section. Each feature will either be coloured green for fully met, amber for partially met and red for not met.

Feature	Justification	Evaluation
🟩 Important information is displayed centrally	Information such as the login buttons and recommendations will be central to the screen. This area will have more space and therefore information does not need to be cramped, improving readability.	This usability feature was fully met throughout my program as the important parts are centrally located. For example, on the login page, the information, buttons, and text field are all central. Furthermore, on the recommendation  Twitter Recommendations Account Settings Authenticate You are not logged in, please press the button below to authenticate and then enter the given pin in the text field below! page, the recommendations, relevant information, and actions are central.  Search bar Recommendations: Refresh Open in Twitter [redacted] This was an effective usability feature was it was easier to use when centred. If it were to one side, it felt unnatural to use and therefore would not provide a good user experience.
🟩 Useful buttons on top-left	According to a survey[4], the top-left corner of the screen gets user's attention first. As a result, important buttons such as the button to open the settings GUI will be placed here, as these are important for the user.	This usability feature was fully met as the useful features such as the search bar and refresh button were located at the top of the screen, reading left to right. This usability feature was effective because Search bar Recommendations: Refresh Open in Twitter [redacted] the eyes were naturally drawn to this position on the screen, and therefore these important functions were visible to the user immediately.
🟥 Expanding boxes	Recommendations will, at first, be short summaries. However, the user can click	This was not met due to time constraints and lack of knowledge of the GUI library. With more time, I could have learnt the

[4] CXL.com website – 10 Useful Findings About How People View Websites

🟥	on a box to expand its size. As a result, the recommendation will have more area on screen and so can display more information. This means that information does not need to be cramped, again improving readability.	library to a greater degree allowing me to create a GUI using this. This usability feature was not implemented and therefore cannot be said to be effective.

🟩 Colour scheme	My program will have two colour schemes, a light and dark mode. Dark mode is beneficial to the user, as it is light text on a dark background. This reduces blue light exposure (which affects your sleep rhythm[5]).	This usability feature was fully met as my application used a dark mode theming, using these colour compositions
		
		`public static final Color BACKGROUND_COLOUR = new Color( rgb: 0x23272A);`
		`public static final Color CONTENT_BACKGROUND_COLOUR = new Color( r: 44,  g: 47,  b: 51);`
		`public static final Color TEXT_COLOUR = Color.WHITE;`
		`public static final Color ERROR_TEXT_COLOUR = Color.RED;`
		This was an effective usability feature as the darker lights are not as harsh as brighter whiter colours may have been on my eyes and the eyes of my stakeholders when testing. None of my stakeholders complained about the dark mode colour scheme.

🟥 Contrasting colours	In settings, an option will allow the user to choose the colour schemes (if light and dark mode are insufficient). This allows the user to customise the program to their needs and to choose colours that are suitable for their needs.	This was not met due to time constraints. With more time, I would have been able to implement this usability feature. This usability feature was not implemented and therefore cannot be said to be effective.

Addressing further development of usability features

🟩	🟩	🟥	🟩	🟥

Some usability features were not met. This is due to a few reasons. However, on the whole 60% of the features mentioned were met.

For expanding boxes, this was not met due to time constraints and lack of knowledge of the GUI library. With more time, I could have learnt the library to a greater degree allowing me to create a GUI using this. In further development, I would have spent more time learning the GUI library in order to create more complex features such as this. The issue with this, is that less information was displayed about the accounts, which limited the usability of my program as stakeholders found it hard to choose using the basic information that my design provided.

[5] Healthline.com website – Is Dark Mode Better for Your Eyes?

For contrasting colours, this was not met due to time constraints. With more time, I would have been able to implement this usability feature. In further development, with more time this could have been coded allowing the user to input HEX/RGB colour values for each of the components that made up my colour scheme and saving this to a file to be persistent. This issue did not seem to limit the usability of my program for my stakeholders, as no one commented on the lack of this feature.

Maintenance issues and limitations of solution

The biggest limitation of the solution is that using the Twitter API, my program is rate limited to only send a certain number of API calls per 15 minutes. If my program exceeds that amount, then it cannot make anymore calls until 15 minutes has passed, and the rate limitation has been removed. This is a very big limitation as if multiple people are using my application at once, then the API calls are going to increase, as a result the rate limit will be hit sooner, and no more recommendations will be able to be generated until this is over. Consequently, this has a negative impact on user experience and overall use of the solution.

The second most significant limitation of the solution is the limit of my knowledge when it comes to the Java GUI library. The time constraint meant that I could not learn this to a degree that would allow me to code the complex features that I had written about in the requirements section of my analysis. As a result, a few of these GUI related requirements were not present. Consequently, this had a negative impact on the solution as the information to help users decide which accounts to follow was extremely lacking which created a worse experience, which was picked up on by my stakeholders during post development testing.

The only issue for maintenance is that of the complexity of certain parts of my program. Because I coded the largest features that used to Twitter API to run asynchronously, it can get a big confusing and complicated when interacting with those parts, as they are running on a different thread and the result may not be when expected. Therefore, for anyone in the future to maintain the solution may have to overcome this challenge at first.

Overcoming limitations and improvements

To overcome the issue with rate limitation by Twitter, I could have setup my own server that caches results from the API, such as account ID's. By doing so, I could use my own caches that would not be rate limited as they would be controlled by me. However, this would be costly to setup and maintain my own servers. Alternatively, Twitter offers a premium for increased rate limitations. To pay a monthly fee, the rate limitations of my application would increase, and I would be able to make more API calls. This would allow my application to support more concurrent users.

To overcome the issue of the GUI library, I could have spent more time learning the library, and if there was no time constraint this would have been a viable solution to overcome this limitation.

Improvements for my solution include:

- Caching retrieved accounts/information on users' local machine to reduce API calls

- View more information about an account when clicking on it, either in a pop-up window or an expanding box
- An improvement I might make to improve maintainability is to undo my decision to make significant API calls run on an async thread. In reality, this only helps for roughly 20 seconds when the program would otherwise freeze and is not worth the trade-off for worse maintainability
- Settings panel to allow user to ignore accounts and interest terms, as well as changing the colour scheme of my program