Introduction

*In Silico* drug discovery is important to find suitable drug targets. The design of these targets is challenging and requires searchable storage to allow identification1. A database can be used to store relevant information about the drugs for future work. Filtering, such as Lipinski’s rule of five, Bioavailability and Lead-likeness, helps identify targets that fulfil certain practical requirements of drugs2-4. Here, a database was created for a given 100 compounds. These filtering criteria were included to help identify likely drug targets. This is beneficial in the drug discovery process as it prevents the misuse of time and money on oral drug targets that are unlikely to be adequately soluble or permeable. From this stage, targets can be identified and further tested to (hopefully) provide an effective drug.

Task 1

# Methods

ChemAxon Marvin plugins were used to calculate the parameters for this database5. Some code was adapted from the code provided by Graham Hamilton’s course material.

# Outline

1. Load the file
2. Delete the old database
3. Calculate the parameters
4. Create the new database
5. Populate the database
6. Apply the filters
7. Show the database
8. Save to file

## 1.Load the File

Open Sd File ctrl-o 
Opening 
Done! User can browse to select a file (.sd) from the computer (Figure 1). The output shows the name of the file selected. Here the user is also prompted to input the number of compounds that will be used in the program (the variable Size in Calculations).

Figure : To load a file (.sd).

## 2.Delete the old Database

Every time the program is run it creates a new database. To prevent an error, the old database must first be deleted. This is done by the user clicking the Start button (Figure 2). After the user has selected the start button, the buttons to calculate parameters, create a database and populate the database become visible. They are then enabled (clickable) after the previous button has been used.

Figure : Start button deletes the old database before the user creates the new one

## 3.Calculate the Parameters

After the user has selected a file, the calculate button is enabled and can be used (it is visible, but not usable, after pressing Start) (Figures 3 and 4). This uses the Calculations class to calculate the parameters by the ChemAxon plugin. The calculations use the file selected by the user. The calculation values are stored in arrays to be later uploaded to the database.

calculate 
create the database 
Upload to Database LogP and LogD had some values that were NaN (not a number). These could not be uploaded to the database. While doing those calculations the program checks to see if the value of LogP (and LogD) is NaN and if it is it changes it to 0.0 (so it can be uploaded to the database). However, as this is a false value, these entries must be deleted from the database. This is done by an ArrayList which stores the index of the values that were changed to be used later. An ArrayList was used as it can have a flexible size for different sets of compounds that may have a different number of LogP values that are NaN. The LogD values are also rounded to three significant figures at this time (to allow a neater table when viewing the database) (Figure 5).

Figure : Calculate button

Calculating 
Calculated! 

Figure : Display seen by user when calculate button is pressed

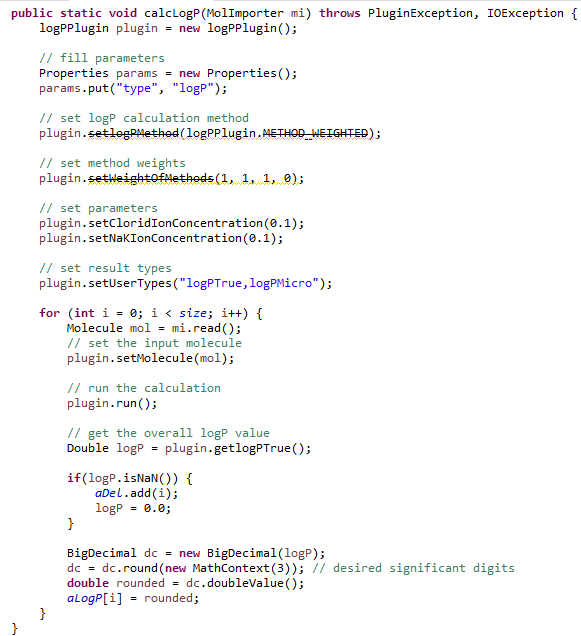


Figure : Code to calculate LogP and save it in an array. Also saves an ArrayList of the positions where LogP is NaN

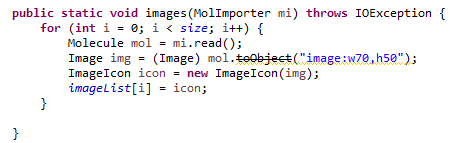
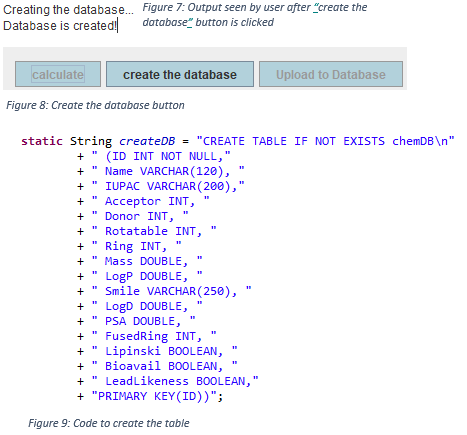
The images are also generated at this time and stored in an array of type ImageIcon (Figure 6).

Figure : Code to generate array of ImageIcons to later display the image of the structure

## 4.Create the Database

The database is created using the button “create the database” (Figures 8 and 9). The SQL query can be seen in Figure 7.



## 5.Populate the Database

Uploading data to databse 
Data has been uploaded! The parameters are uploaded to the database using the arrays made earlier (Figures 10 and 11).

Figure : Output to user after upload to database button is clicked

Figure : Upload to Database button

A parameterised query is used (Figure 12) that allows the user to loop through the arrays and add each compound calculations (Figure 13). Each compound is assigned an ID (primary key) when it is added to the database. These start from 0 and correspond to the original the array index.

Figure : Query to insert to database

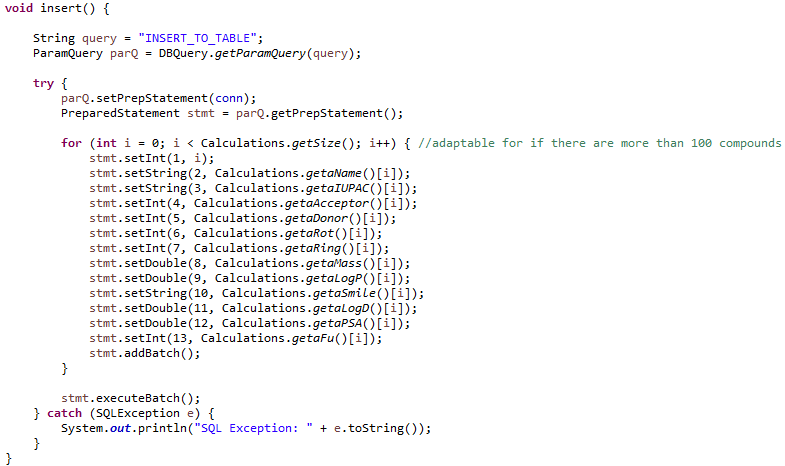


Figure : Inserts the data to the database from the arrays made earlier

At this point, the false logD and logP entries are deleted from the database. It deletes these rows using the ArrayList made previously (aDel) (Figures 14 and 15). These IDs match the index in the array allowing the correct entries to be deleted. A parameterised query is used based on the ID’s and the numbers stored in the aDel ArrayList.



## 6.Apply the Filters

filter There are three filters possible in this program: Lipinski, Lead-likeness and Bioavailability. The filter button checks which compounds pass the filter and then updates the relevant entry to true in the database for that filter (Figures 16, 18 and 20).

Figure : Button to add the filters to the database

Input 
Please enter a mass for Lipisnki filtering Lipinski’s rule of five can be run using different numbers for the filters. This adds a level of adaptability to the user. The user is asked the numbers they wish to use for the four different filtering criteria (Figure 17).

Figure : Query to add Lipinski as true to the database for the relevant compounds

Figure : Code to add Lipinski filtering to the database

Figure : Allows the user to input numbers for Lipinski filtering. Example shown here for mass



Figure : To add Lead-likeness to the database. Always uses the same numbers unlike Lipinski where the user can input the numbers they wish to use for the filter.

Both Lipinski and Lead-likeness use the values from the database to calculate the parameters (Figure 19). Bioavailability, however, is filtered based on the arrays created when the calculations are done (Figures 21-23). This is because there was limited time to work on the SQL query. Future work could add this feature.

Figure : To pass the filter, 6 out of the 7 criteria must be fulfilled. This is difficiult to achiveve in an SQL query so was done here using Java with the previous arrays. The IDs that pass the filter are stored in an arrayList (due to the flexible number of compounds that may pass)

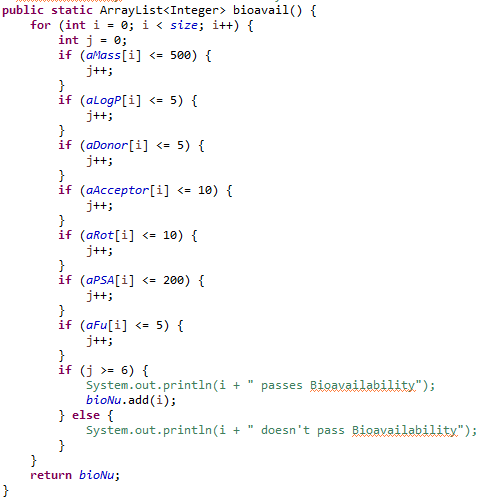
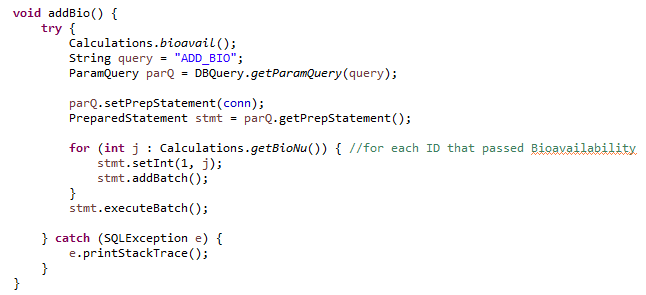


Figure : Query to add Bioavailibility to the database for each ID that passes the filter

Figure : The arrayList is used to run the query for each ID that passes the filter (similar to Lipinski and Lead-likness)

## 7.Show the Database

There are options to show the database without the filters, with the filters, or show only the compounds that pass a selected filter. Each button runs the associated query using only the information from the database to populate the table (Figure 24)



Figure : Buttons available for the user to show different versions of the database in different tables. Can also save the selected filter to file

**Show Table Without Filters**

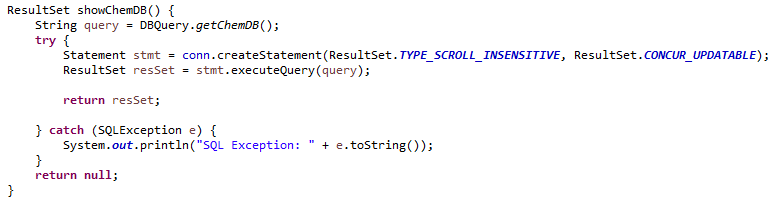
Shows the table without the columns for the different filters. Allows the user to view the compounds. Can click on column headers to sort by that column. The default order is by ID. A query is used to get the information from the database that is then added to a table and shown to the user (Figures 25-28).

Figure : Query to retrieve the data from the database

Figure : Saves the data in a resultset

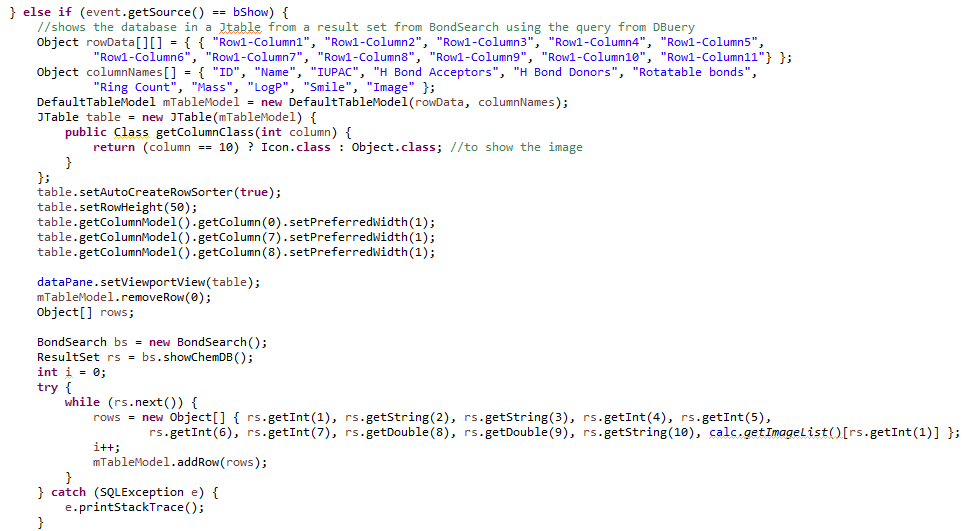


Figure : Code to display the resultset in a table. Also, includes the code to show an image of the structure in the table

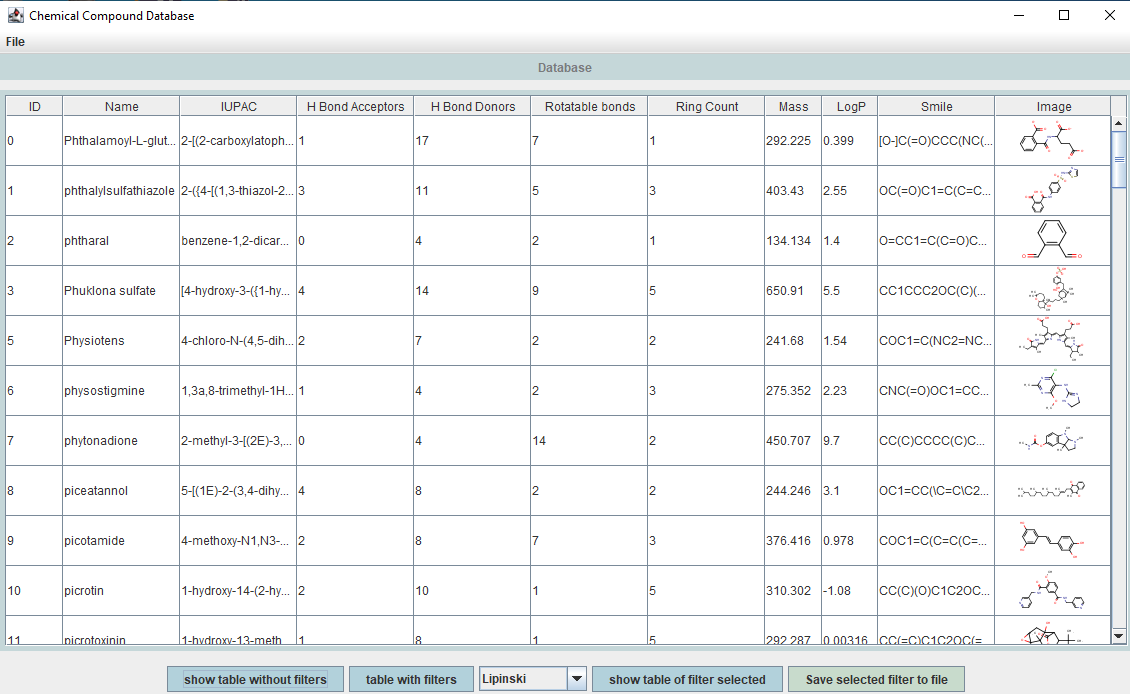
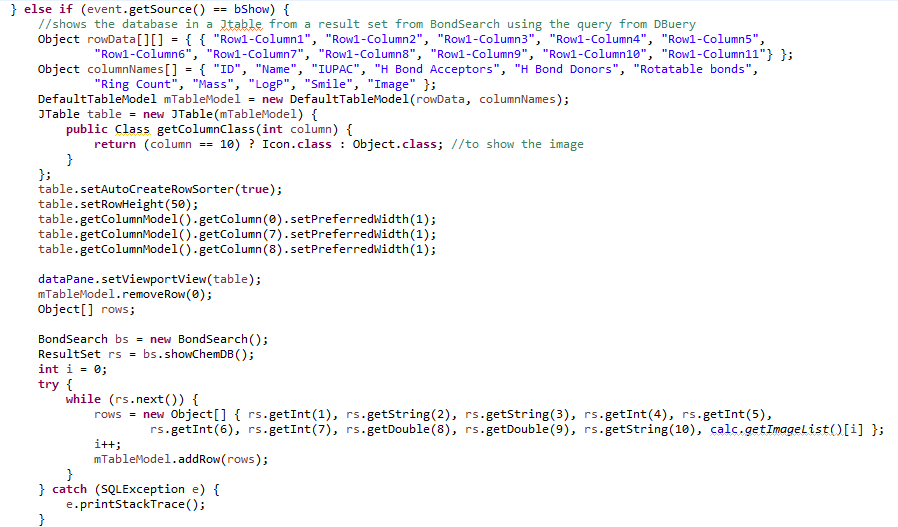


Figure : Table without filters

**Show Table With Filters**

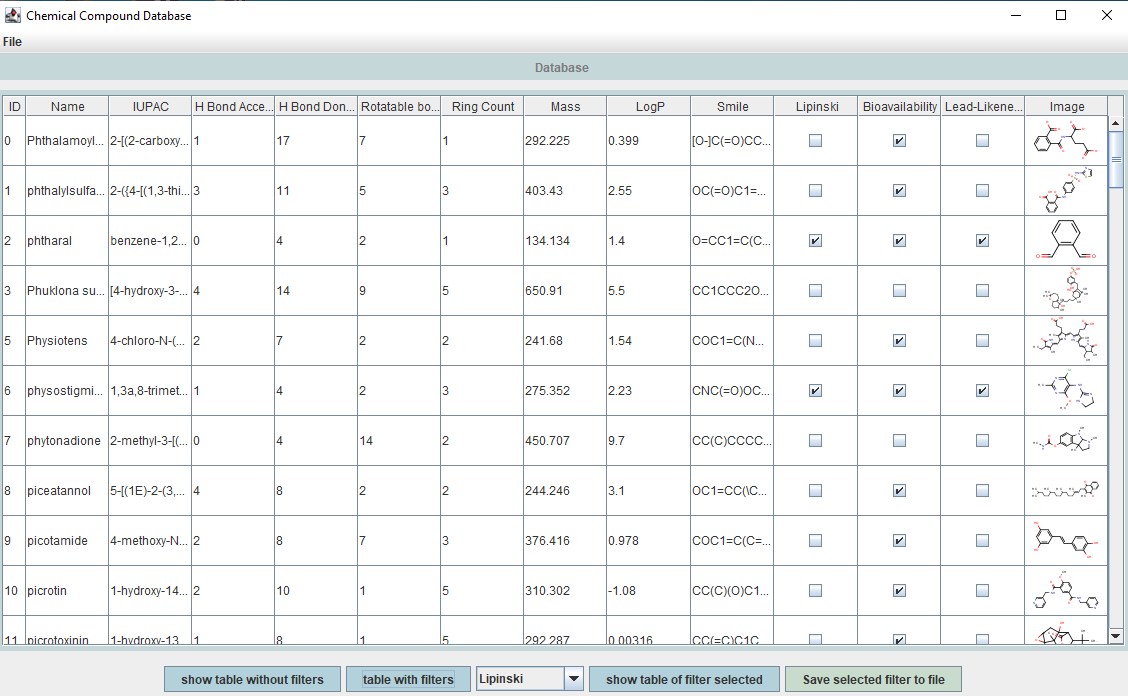
Shows if the compound passes the filter with a tick. This shows that more compounds pass Bioavailability than Lipinski and Lead-likeness. A different query is used here to also get these columns for the filters (Figures 29 and 30).

Figure : Query to show the table with filters

Figure : Table with filters



**Show Table of Filter Selected**

Select a filter from the list and click the button (Figures 31-34). This shows all the compounds that pass this selected filter (= true).

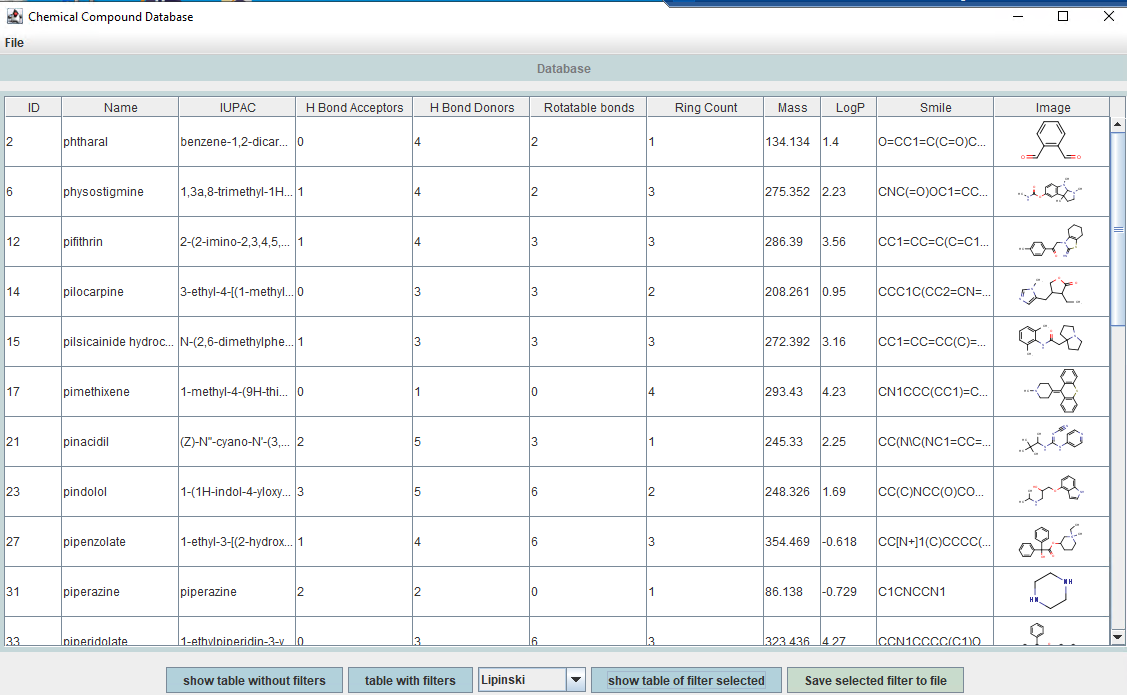


Figure : Query to show only the compounds that pass Lipinski filter (i.e. where Lipinski is true)

Figure : Table of compounds that pass Lipinski

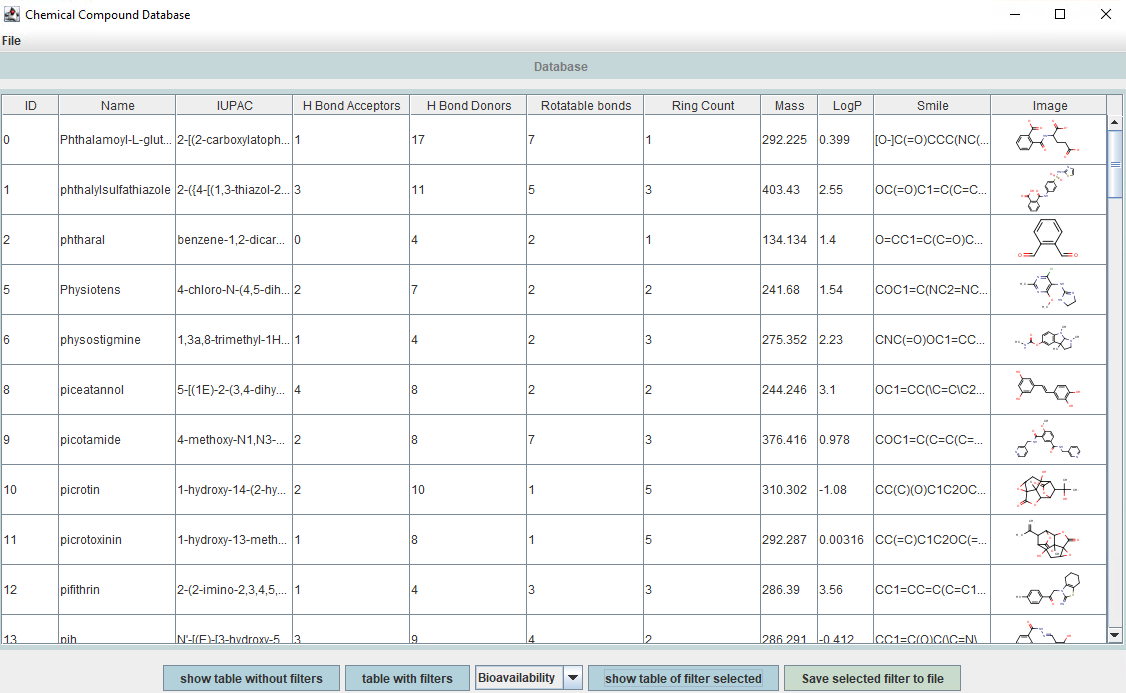
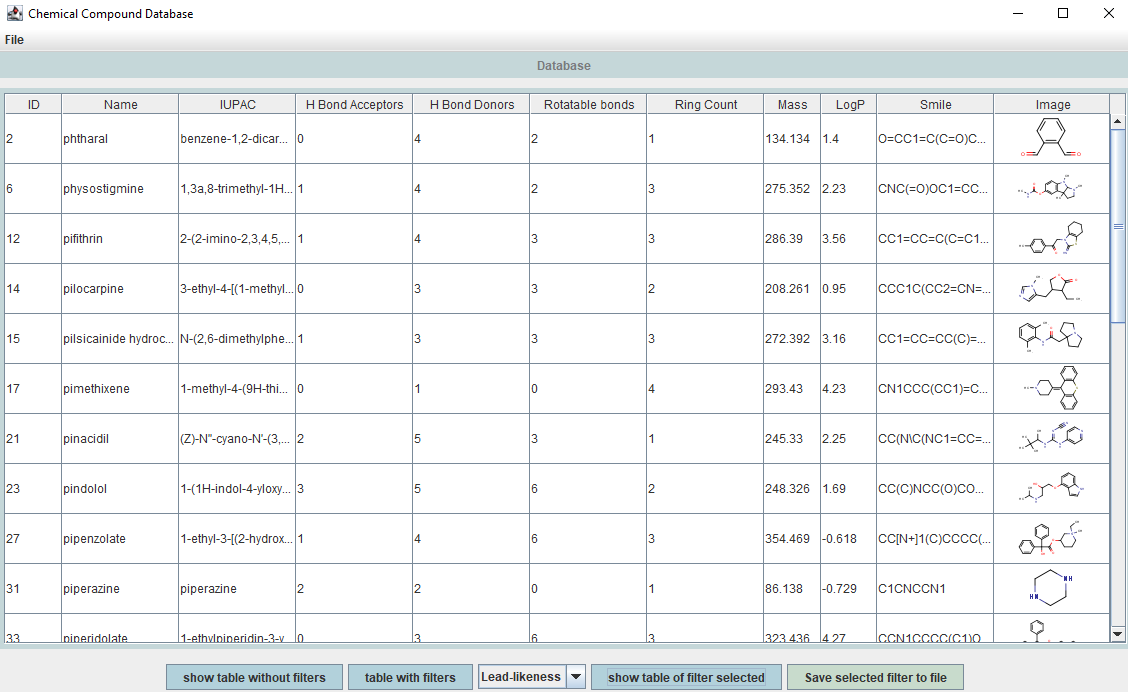


Figure :Table of compounds that pass Lead-Likeness

Figure : Table of compounds that pass Bioavailability

## 8.Save to File

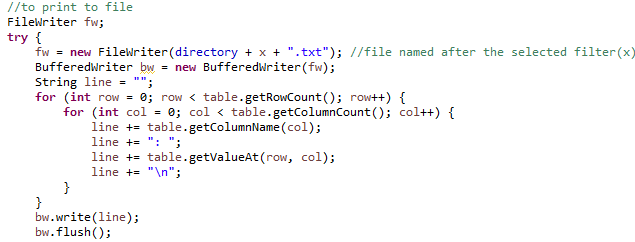
User can select a filter using the list and click the button to save the compounds that pass that filter to a file (Figures 35 and 36). Each file is named after the filter. The file is saved in the same directory as the one used to open the file at the start. This means the program can be used on other computers.

Figure : Code to save the compounds that pass the selected filter to a file

Figure : Button to save compounds that pass the selected filter to a file

## Adaptability

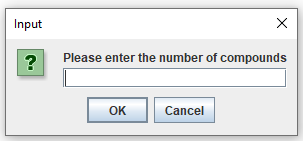
This program was tested using 100 compounds. Some changes were made that would allow more compounds to be used. The user is asked to input the number of compounds they wish to use (Figure 37). This will then decide the size of the arrays, the number of times the loop is repeated to calculate the parameters and the number of entries inserted into the database.

Figure : Allows user to input the number of compounds

Everything else in the program was created so that it does not depend on a number. For example, ArrayLists were used to keep track of LogP NaN and the compounds that pass the Bioavailability filter.

An issue may occur when presenting the tables to the user. Currently, all 100 compounds can be easily viewed. A JTable, however, is limited to 100 rows; a database with more compounds would require a feature that would allow more than one table with the user able to click between them (with 100 entries per table). It would be beneficial to let the user know how many tables there are and what table they are currently on.

It was considered to add a search feature to the program. However, given the small number of compounds and the existing ability to sort these by name and ID, it was determined not to be necessary. In a database with more compounds, this would add some functionality and help in the cases where there are multiple tables required.

## Limitations/Next Steps

The main limitation of this project was time. With more time, changes would be possible. For example, deciding which compounds pass the Bioavailability filter using the database instead of the arrays.

Another element that could be relevant to the user is showing the number of compounds that pass a filter when the table is shown. Future work would perhaps include this feature.

Finally, the database always uses the same login and in a future program, there could instead be an option to allow the user to login and choose where they want the database to be saved. This was not deemed necessary for this current project.

Overall, these changes were not a necessity given the time frame but could be incorporated in future iterations of the project.

References

1. Rifaioglu AS, Atas H, Martin MJ, Cetin-Atalay R, Atalay V, Doğan T. Recent applications of deep learning and machine intelligence on in silico drug discovery: methods, tools and databases. Briefings in Bioinformatics. 2019;20(5):1878-912.

2. Benet LZ, Hosey CM, Ursu O, Oprea TI. BDDCS, the Rule of 5 and drugability. Adv Drug Deliv Rev. 2016;101:89-98.

3. Colomer I, Empson CJ, Craven P, Owen Z, Doveston RG, Churcher I, et al. A divergent synthetic approach to diverse molecular scaffolds: assessment of lead-likeness using LLAMA, an open-access computational tool. Chemical Communications. 2016;52(45):7209-12.

4. Hinderliter P, Saghir SA. Pharmacokinetics. In: Wexler P, editor. Encyclopedia of Toxicology (Third Edition). Oxford: Academic Press; 2014. p. 849-55.

5. Marvin 21.4, 2021, ChemAxon (<http://www.chemaxon.com>)