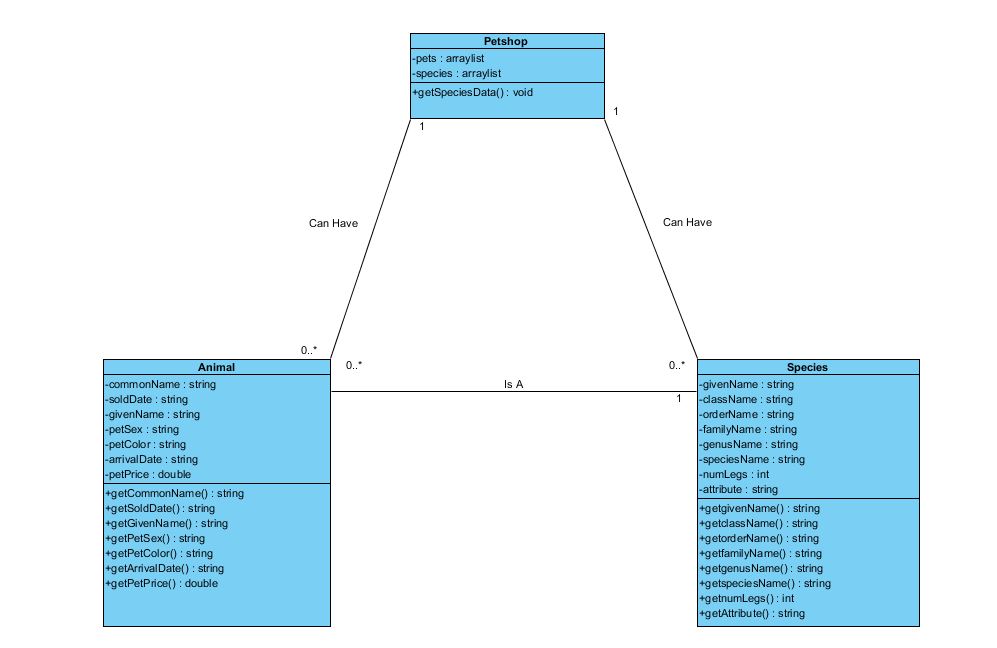
Object-Oriented Programming Coursework Report

**Requirements**

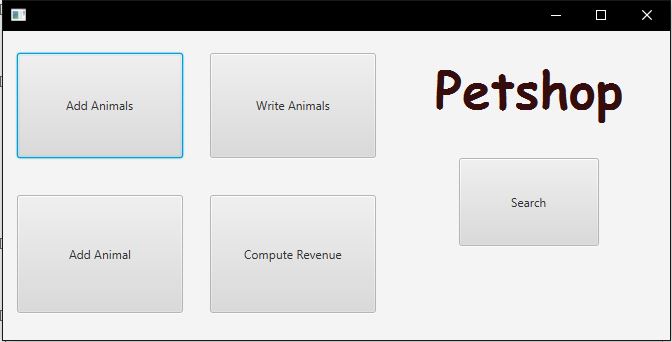
1. Develop an administration tool for a pet shop in Java.
2. The shop has to buy and sell pets.
3. Each pet is of a certain species.
4. Pets are categorized using a reduced biological taxonomy.
5. Some pets have additional attributes (e.g. “talking”).

**Design**

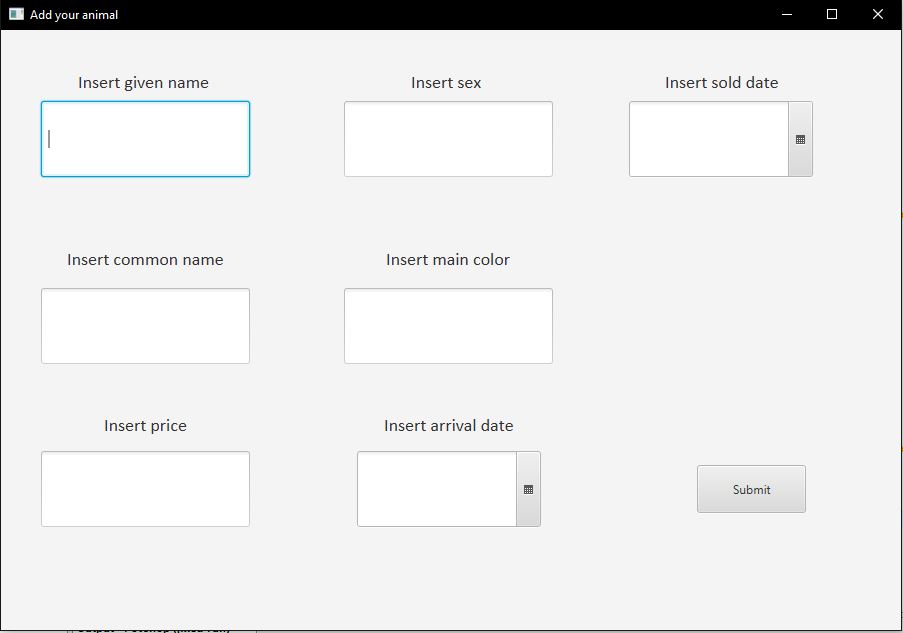


My Petshop application will consist of three classes (Petshop, Animal and Species). Petshop will be the main class that will contain two arraylists (pets and species). These will be very important as these arrays should have the objects used in the program. When looking at the two classes, Animal and Species, it can be noticed that there is only one common attribute between them. That is the common name/given name and they will play an important role in the implementation stage of the application when it comes to the search function. As for the attributes most are of type String which might come across as strange for attributes such as arrival date and sold date. However, this is due to the fact that the dates are made up on digits and “-“and a string type will allow for better manipulation and results. Another interesting attribute is petPrice which has a type double to allow for decimal points. A design decision I came up with was instead of adding separate classes to pets with special attributes, is to create a method attribute and assign it to species. In this way the parrots and reptiles in the application will be assigned their attribute just in that class. This was designed to simplify the application and not make extra attributes complex.

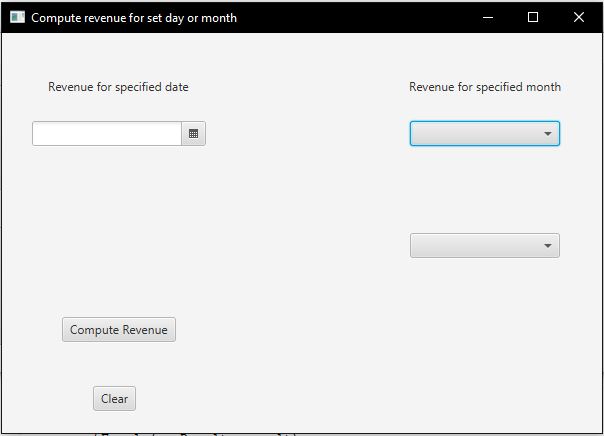
**GUI (Graphic User Interface)**



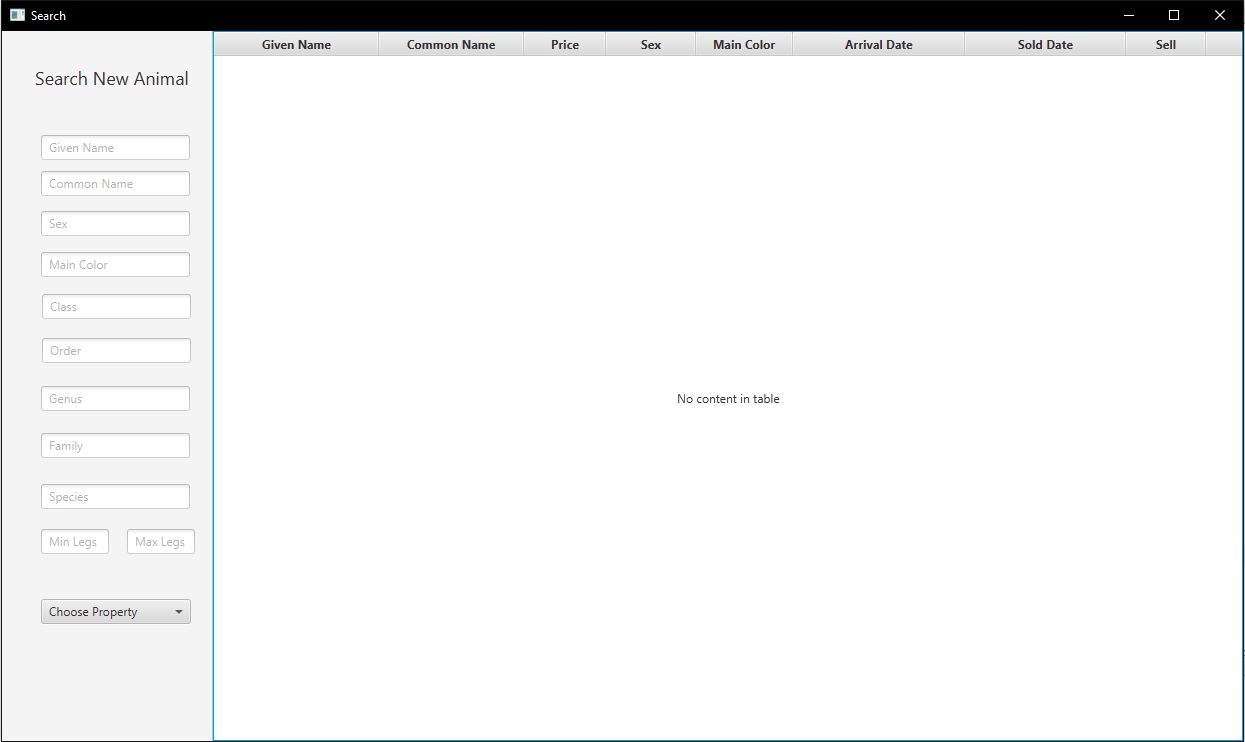
For the design of my GUI I used JavaFX which was the most intuitive way to structure my interface. I chose JavaFX for its ability to easily layout the GUI with necessary buttons labels and more. Another advantage was the ability of adding Id’s to the buttons and also pre-applying the names for functions they will run. This is very useful and will aid in the implementation phase so that each function is assigned to specific buttons. For the main screen once the application is launched I used a scene with 5 buttons representing the functions that had to be implemented. This is a simple yet effective design that should allow for users to easily traverse the program. One function that doesn’t have its own button is “sell animal”. The decision for that was changed by having the sell animal function be part of the search function as will be shown later. Another design decision was having the top buttons be the ones that require a file to be chosen. This was made so that the user intuitively adds data to the program which can be then later manipulated using the rest of the buttons. The other three buttons should move the user to a new scene.



The add animal scene in the GUI was designed very simply. A use of multiple text fields for each attribute makes it easier to code in the implementation phase. There is no cluster and it makes it clear what attributes need to be assigned where. A very useful and interesting feature in JavaFX is the date picker which allows the user to select date from a calendar. This makes the design much smoother and stops the user from having to input the dates themselves in form of a string. This design also should allow for easing formatting of the date as retrieving the value from the date picker and then reformatting it in whatever way necessary is a big plus.



The compute revenue scene had two extra addition in its design. The combo box which will allow to hardcode the months and implement the years without using the date picker which in this case would be a poor design option as we do not need a specific date for that part of the functionality. A label will also be used to directly display the results on the scene.



The search scene is designed to have two distinct sections. The left section is where the main searching and filtering takes place and should allow the user to filter on key presses which is enabled in JavaFX features. The big section to the right will show the results of the search under the table headers. A very important column is Sell which will after implementation show a sell button giving the opportunity to use the sell function within the search scene, saving space and making the design a bit more proper.

**Implementation**

The first step in the implementation phase was to implement the getters and setters for the species and animal classes, as designed in the class diagram. In the main class we set up the arraylists for species and animals as well as hardcoding the species so that we can use them for the search function.

The first major function used in the application is add animals. This functions allows to add a list of animals provided in a file. The add animals function uses a bufferedreader and filereader imports to read a file chosen by the user. This is done using a while loop that goes through the selected file and creates and array of length 5 which is enough to support all the attributes needed for an animal. I used a triple recursive constructor which basically meant that an animal could lack an arrival date, sold date or both and still be read and printed. This is done by first adding the attributes up until arrival date and adding those to an array. The code will then loop and check if the array is longer as it was previously split at the comma position using split function. If the array’s length is greater than 5 then the loop will continue and add the correct attributes. This will result in the animals added to be sorted into neat lines and no errors arise when it comes to some animals having extra attributes.

The write animals function is mainly used after user adds a single or multiple animal first. Its purpose is to output a file of all animals grouped into sold and unsold. The first step is to check whether the file we are writing to is null. Using a simple if statement we avert running into that problem. Using a variation of previously mentioned imports we use a comparator to compare the animals sold and arrival date. This will sort the animal depending on their arrival and selling dates. After the sort is done a for loop is used on the sorted list and the data is written and outputted onto the file by “writing” each attribute in order into it.

The next functions use GUI scenes as shown previously and had more interesting steps that had to be implemented. The first of these functions is add animal which allows the user to add a single animal with given parameters. The first step in this function is to validate whether the user has put any input in the text fields which is done through an if statement and isEmpty() method which returns nothing if user had not entered text in the fields. The next steps involve creating a new scene when the add animal button is pressed. Some formatting had to be implemented on top for the arrival and sold date. This was done using a date time formatter import which allowed to change the date format received from the date picker. After all the data has been retrieved from user the function adding them to the pets arraylist and closes the window.

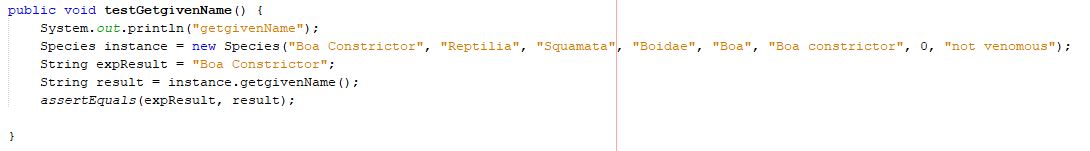
Compute revenue computes and outputs the revenue of the specified day or month, that is the sum of the selling prices of the animals sold on that day or in that month. For the combo box that contains the months, those were hardcoded in an array. As for the year an array of length 100 was created and made too print all the years from current one back 100 times. This then displays as the months and years needed for function to work in specified month. A reset button also added which simply set value of all buttons to null. The revenue itself is calculated by filtering out the prices from the pets arraylist then summing them for a particular date. For the specified month the value was taken off the user then the prices were filtered by month so that the sum is done for all animals that contained a sold date in that selected month.

The search function contains both the sell function and search options. Their purposes were to output all the animals including the use of filters. The sell function changed the status of the sell animals to sold and outputted the current date. For the search function the initial method used is common search which filtered animals by setting the printed property to lowercase and checking if it is contained in the attribute itself. If the result is true then it is printed in the columns. This was done for all attributes of species and animal. The ones that required some special attention were the dates which just used the previously mentioned date formatters.

The sell function operates by the user pressing a sell button corresponding to an animal. This triggers an on event action and changes the button to sold. An added feature is that after an animal is sold the text of the button changes likewise and the button is disabled. In cases for animals that are already sold the button automatically assumes that state. Otherwise when an animal is sold we use a local date import to get the value of the current date and output it on the sell date column.

**Testing**

For testing I used the IDE’s proposed testing method. Using that I was able to run tests for setters and getters.



As shown above, using test variables and expected results, the application is able to run tests on this getter. This was done multiple times on various attributes and methods. A more complex test could be run on the function however these were done during implementation as validation. Tests such as checking what happens when the user does not input anything in the text fields were done and then results allowed for changes to be made to work round the errors.

**Conclusion**

To use the application the user needs to first select button that adds data to the program to interpret. This is done using add animals or add animal. Using add animals requires simply to insert a text file with all the pets and their respective attributes. On the other hand, add animal puts the user through a new window where you are permitted to add all the attributes yourself. After pressing submit the window is exited and the data will be inputted into the application. To use write file the user simply needs to select a text file to which he wants to add the previously added animal/animals to. The compute revenue works by selecting either a specified date and computing revenue for that or a month and year from the combo boxes to check revenue for a month. Finally the search works the intuitively by having a left side filter which works automatically and updates on every change. To sell an animal user has to simply press the sell button which will deactivate and be names sold.

The drawbacks of this application is that if a user wants to add an animal outside the hardcoded species the application wouldn’t work. This is quite difficult to achieve as there is a massive number of different species of animals and they can’t be all hardcoded. This can be averted if a database was used and integrated with the application. Another drawback is the fact that for adding animals the text file inputted has to have all the data formatted in very specific way. This means that any errors in the text file imported leads to malfunction of the application in reading the data. This can be averted by having some sort of method to auto-sort the text files or check on their validity using a database. A drawback of the sell function is that once a pet is sold there is no way to reset the decision except by restarting the entire application which isn’t ideal. There is a possibility of adding a reset sell button to the GUI however that doesn’t necessarily mean the application will improve from it. Finally, in the add animal there is a possibility of the user to add a sold date earlier then the arrival date which would be contradictory. This is a validation issue and can be improved by comparing the sell date to the arrival date and checking if its greater or smaller then notifying the user to change the sell date if its earlier then the arrival date.