**Completed features:**

Add individual animals with given name, common name, value, sex, main colour, with optional purchase and selling date entry (if no purchase date entered, the program uses the current date instead)

Add a list of multiple animals from a txt file in the format given in the coursework specification

Sell an animal from a displayed list of all animals stored in the program, with an optional sale date entry that defaults to the current date. If an animal is already marked as sold, the program will alert the user

Write a list of all animals to a txt file. The list groups all sold and unsold animals together, although does not do any sorting by date

Search through the list of saved animals, with all required filter criteria implemented. The user can apply as many or as few of the criteria as is required and the search will still function

Computation of revenue for both specific days and specific months is implemented

The program will automatically apply the discounted price brackets defined in the specification to all animals that are input, be that manual add or adding through a txt file

**Program operation:**

The individual features of the program are split into their own tabs for easy navigation.

Add Animal: to add a single animal, type in given name, select the common name from the drop down box, type in price (takes decimals), pick sex from the radio buttons and type main colour. Furthermore, if an arrival and/or selling date are to be entered, they must be input in the format YYYY-MM-DD. Once all data is input, press submit

Add Multiple Animals: press the browse files button to navigate to and select the .txt input file. After that, just press submit

Sell Animal: on the left of the window is a list of all currently stored animals. Select the animal to sell, and either leave sale date blank to store the current date, or input a date manually, again in YYYY-MM-DD format.

Write Animals: first navigate to the desired destination folder with the choose file destination button, then enter a file name, and then press write to file

Search Animals: All the search criteria on this tab are optional. The given name box takes any string, while the leg count boxes take integer inputs. The classification dropdowns are not linked to each other (i.e. selecting Mammalia in the class dropdown does not filter the entries in the order dropdown to only Mammalia). Once all required criteria are set, the search button will generate a list of matches. The reset filters button returns all filters to null.

Compute Revenue: to compute the revenue for a specific date, enter the date into the left side box, in format YYYY-MM-DD. After submission the result will be displayed below. To compute a monthly revenue, enter the month into the right-side box, in format YYYY-MM. After submission this result will also be displayed below its respective entry box.

**Design Decisions:**

When initially designing the object orientation of the program, I considered fully representing all values for all parts of the reduced classification tree in the form of abstract classes. While this would have been the most accurate approach, and would have prevented the need to manually enter the attributes into each individual animal class, I soon realised that to do so would take much longer than if I were to simply enter all classification attributes into each animal class individually. However, this method would not really count as object orientation, so I decided to compromise between the two approaches and flesh out part of the classification tree, that being the Class level as well as the parrot order, and manually enter the rest of the attributes.

For the layout of the UI, my initial design involved a main menu window, which had buttons that opened separate windows for each function of the program. After beginning to implement this however, I realised that it was an unnecessarily complex solution which involved passing data between the windows quite a lot. After some thought, I decided to settle with a single window which contains a separate tab for each function. This way data such as the ArrayList of the animals was easily accessible to all function without having to create get methods.

When creating the single animal add function, my design started with a single text entry field which would have involved the user manually typing all attributes into the field, in the entry format defined in the specification. Obviously, that would have been far from user friendly and would have left the process easily prone to human error. For those reasons, I split the entry up so that each attribute has its own field, and kept the manual text entry to a minimum with a dropdown menu for the common name and radio buttons for the sex. I would have liked to implement a popup calendar to prevent the need for manual date entry in the correct format, but was unable to realise that idea.

The hardest function to implement, in my experience, was the search function. After considering the issue, I decided that the best way to implement it was to filter an already full list instead of searching through one list and adding to another empty one. My initial approach to this method was to create a clone of the animal list and have an if statement for each criterion, where each statement would iterate through the list and use the .remove method to remove any animals that didn’t fit the criteria. While this implementation did display results, it was often showing anomalies that did not fit the criteria. Eventually I replaced the .remove statements with .removeIf() statements, where each one had a predicate logic statement inside. This proved to be accurate and returned the correct results.

**Testing/Evaluation:**

The approach I took to this program was to develop one function at a time, testing along the way. The general approach I took to testing was to first make sure that valid inputs returned correct outputs, and then where applicable I did boundary testing on input fields. For example, in writing the function that handled discounting the animal prices after they’d been in the shop for certain periods of time, I found after boundary testing that my initial code was in error as I had only compared the day and month of each date. As a result, if an animal’s arrival date was in a previous year, the comparison did not take that into account and often resulted in an incorrect result.

**Please see the attached jpg file for the class diagram**