# Agile Coding Team Lab Assignment 2

*Lachlan Copsey (Task 1) 30333504*
*Nathan Blaney (Task 2) 30331062*
*Nine Hall (Task 3) 30332017*
https://github.com/ITECH2306FedUni/TeamLabAssignment

## System Overview

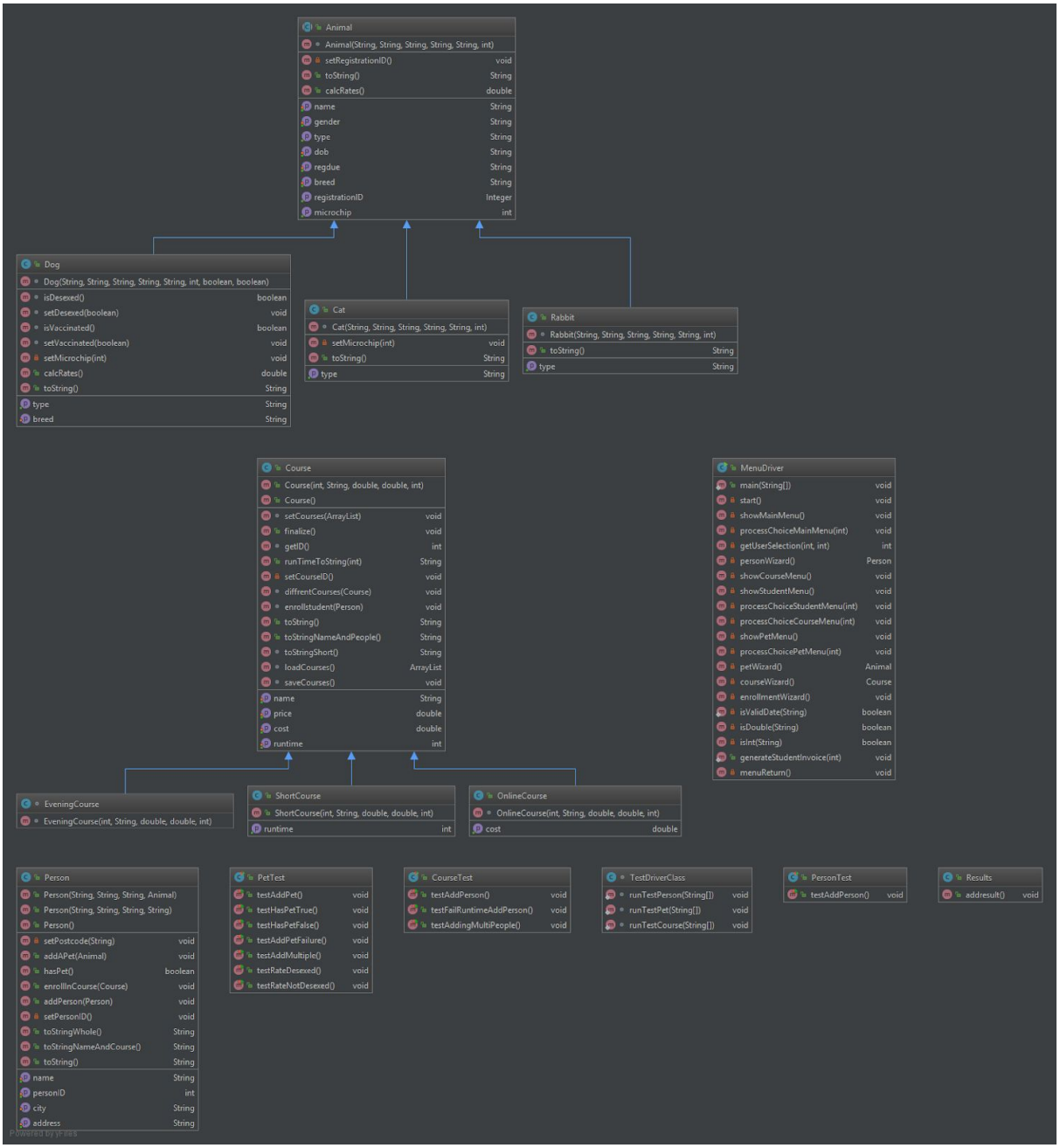
This system further implements our fictional ratepayer tracking system from Assignment 1, it tracks a list of ratepayers, their pets, and their enrollment in local adult education classes. The system is implemented in two parts: a set of independent classes that implement the data and methods for the different objects to be tracked and a front-end MenuDriver class that provides a Command line interface to class methods and data. We chose to forego the option to import a clean project from the academic staff as we had already designed our original solution with extensibility in mind.

The system can be used to store a list of ratepayers, and their details, as well as register and check the details of courses, enroll ratepayers into a course, and track their pets and their registration details. It also can be used to calculate rates, collect and check the vaccination and desexed status of animals, track pet microchips. The system contains various systems to validate all records during entry, as well as checking formats for registration dates and dates of birth for registered pets and citizens the system checks for compliance to regulations ensuring restricted dog breeds are not being registered in the city and .

Our software uses the command line as well as file input output for data storage and retrieval this allows for manual data entry as well as loading data from disk. The system can additionally generate formatted reports for Enrollment in courses, Financial reports including expenditure and earnings for community classes, and Pet Registration Invoices as well as generating course results reports for students.

The system can manage and control students and courses, including being able to create and delete courses and students, this also opened up to a full admin tools for a teacher to use like adding in results for a student and their courses as well as being able to save a student and all of their courses back in.

# Class Diagram



# Testing

Using Junit we were able to design and implement a set of unit tests for our solution, we also combined this with Gradle for build configuration and Jenkins for continuous integration. This is a similar approach to Assignment One in which we used Jenkins and Junit to automate the majority of our testing, tests were run for every single build and test reports were published. We used the same Test classes and simply expanded them to cover the new implementations for Assignment 2.

To validate our application from the user interface we use a common Test Runner class (TestDriverClass) that calls each individual unit test for each class, we did this to keep the Test system simple while providing a structure for our testing suite to grow further, and expand.

# File format

We used three different file formats for the application, plain utf-8 text documents with each datapoint on a new line. It might have been better to collaborate on one file format but that comes with a number of challenges: writing to the same file, partial updates, and increased strain on developers. This solution has a number of limitations, some features had to be cut like storing and reporting the names of students in course reports because the state of personal and student records can become desynchronized and records can be unpopulated when querying them for different reports.

## Courses.txt

Course data is stored in courses.txt, it is a *very* simple file format. Here's an annotated record.

```
Course                //Course Class
1                     //Course ID
Spider Husbandry      //Course title
300.0                 //Cost to run
14.0                  //Price for students
6                     //Runtime (number of sessions)
39                    //Number of enrollments
```

The load method will break on a corrupt record, show an error, and drop loading the file.

## Course Expenditure Reports

Initially expenditure reports were supposed to be formatted in pretty print, but this was scaled back to csv because it sidestepped a large number of alignment issues and has much greater utility for data and analytics because it is more easily imported into

spreadsheet and database applications. The format should be fairly self explanatory. But briefly, the first record is the key, the last record is the totals and all the intervening records show individual courses.

```
course, expenditure, price, students, earnings
Spider Husbandry, 300.0, 14.0, 39, 546.0
Z-Fighting, 9000.0, 20.0, 4, 80.0
Spending:, 9300.0, Earnings:, 626.0, Profits:, -8674.0
```

## PersonAndPetData.txt

Person and Pet data is stored in PersonAndPetData.txt, it follows a basic CSV layout
Index 0 on each line determines what the record corresponds to
Below is an example of the file

```
Person,Nathan Blaney,31 Nowhere Street,3977,Casey
Cat,Dim Sim,Tissue,F,05 01 1998,05 01 1998,14
Dog,Pug,Fido,F,05 01 1998,05 01 1998,13,false,false
Person,Lachlan Copsey,69 Rangeless Drive,3977,Casey
Rabbit,Floppy,Fluffy,M,05 01 1998,05 01 1998,0
Person,Nine Hall,56 Torvald Court,3977,Casey
```

## RegoInvoiceNathanBlaney.txt

RegoInvoiceNathanBlaney.txt represents the Rego invoice for the rate payer Nathan Blaney, each rate payer has their own file which is generated in the program, the file name is RegoInvoice + their full name.
Below is an example of the file

```
Nathan Blaney has 2 pet(s)
Tissue the Dim Sim, a type of Cat
Tissue was first registered: 05 01 1998
The rate to pay is: 16.36
Fido the Pug, a type of Dog
Fido was first registered: 05 01 1998
The rate to pay is: 21.76
Total rates to pay is 38.11
```

## StudentInvoice.txt

This is the student invoice file format, everytime a invoice is generated it creates a file called StudentInvoice.txt this invoice displays all of the currently enrolled courses and all of its information on the course and that last part is a final cost for all of the students courses .

```
### Course Invoice For Nathan Blaney ###
Course ID: 1 Course Name: Drifting 101 Course Price: $69.0 Course Runtime: 2 Course Type:
class lab2ass.Course
Course ID: 2 Course Name: Homework Course Price: $23.0 Course Runtime: 3 Course Type: class
lab2ass.ShortCourse
```

```
Course ID: 3 Course Name: Sleeping Course Price: $56.0 Course Runtime: 5 Course Type: class
lab2ass.EveningCourse
Course ID: 4 Course Name: Java 4 Nerds Course Price: $68.0 Course Runtime: 7 Course Type:
class lab2ass.OnlineCourse
Final Total: $216.0
```

### StudentSave.txt

This is the save for the student data, this works by saving the name of the student and all of the courses names, the word “Break” in between tells the program that there is a new course below it to add on this makes it so when the scanner sees the line it knows the next line it will enroll and search through the course list for the name then if it finds it it then adds it to the student by searching its name.

```
Nathan Blaney
Break
Drifting 101
Break
Java 4 Nerds
```

# Individual Questions

## Nine

### Polymorphism:

I subclass course to override specific methods for online courses (which have no operating costs for the council once set up), Short courses (which are offered only as 3 day courses) and Evening courses which have no real differences in their representation, but should be differentiated nonetheless. By overriding the getters and setters for these properties I can enforce limits and ensure they can be accounted for while generating reports and making calculations. This is a lot cleaner than having to write switches or use enumerated types for different types of courses. This enforces a consistent interface to the class, because while iterating through the ArrayList that stores courses in memory the same getters and setters can be called on each element, even though this accesses class specific implementations of these methods.

### Exception Handling:

There are multiple places in which exceptions are generated and they are handled in different ways, in my loading method a try catch block is used to return a specific error message when an exception is reached. In MenuDriver I implemented validation using a wrapper method that uses exception handling to determine whether to return true or false

depending on whether the input line can be correctly parsed. This is more useful for iteration than standard java exceptions.

## Static Member (Object Orientation):

To avoid writing a manager class we made a static ArrayList that contains a list of all courses that are instantiated, this static member exists in memory for as long as there is a single course. To prevent the static member from being deleted MenuDriver instantiates one from the default constructor which doesn't get an ID or added to the ArrayList, so when we create a course with properties using our overloaded constructor it gets the 0 index. This has a number of advantages over a manager class, most importantly that information about all instances of the class can be accessed from within an instance, which we don't currently use, but would be useful in the real world because a method could be called on a single record to, for example, calculate a weighted mark, or indicate the percentage of all students that enrolled in it. It also seemed silly to write a wrapper to ArrayList when we're already wrapping exceptions and constructors all over our codebase, language features should be leveraged to write terser and simpler code.

# Blaney

## Polymorphism

I subclass Animal to override specific methods for each Pet type, each Pet type has their own toString method which covers the specific fields relating to that animal.
Dog's have their own calculation of rates method to handle whether a Dog is desexed or not and the difference of rate.
By overriding the getters and setters for a few cases such as setBreed in the Dog class I can check the breed against a list of Restricted Breeds and throw and error if the breed is on the list.

## Exception Handling

There are multiple places in which exceptions are generated and they are handled in different ways, in my loading method a try catch block is used to return an error message. In MenuDriver I implemented validation using a wrapper that uses exception handling to return a boolean.

## Object Orientation (Encapsulation)

I used Encapsulation primary in the Animal and the Pet subclasses, using Encapsulation and Data hiding variables in these classes are set as class private and can only be accessed through specific getters and setters when needed. Encapsulation keeps the methods and variables of a class are hidden and safe, because variables cannot be accessed directly data validation can also be implemented through getters and setters.

# Copsey

## Polymorphism

When pulling information for courses and person i was able to get the otherriden getters and setters to extract the information that i needed from the different classes, like i had to pull information about the courses and they all had different subclasses and being able to still access them the same. As well as the other items that i had to use getters and setters to just pull the information that i needed without having to hard code them into the program which then makes the program more or less modular in the future with expending all of the sub classes

## Exception Handling

Throughout the project there was many Exception handlers, but one that was key for me during this assignment was validation of arrays and keying in the index, as well as making sure that the file that the user is going to save and load does / does not exist, this made it easy to see and to stop error from happening.

## Object Orientation

In this assignment i used an arraylist to store information on people / students, this was a master list that could create a person object of itself , this was key to make sure that i would have to create heaps of person object or have a person / course manager as the mother class would handle all of the smaller lists, this was useful as i didnt need a wrapper to another manger but all i had to do was point to the static motherclass.