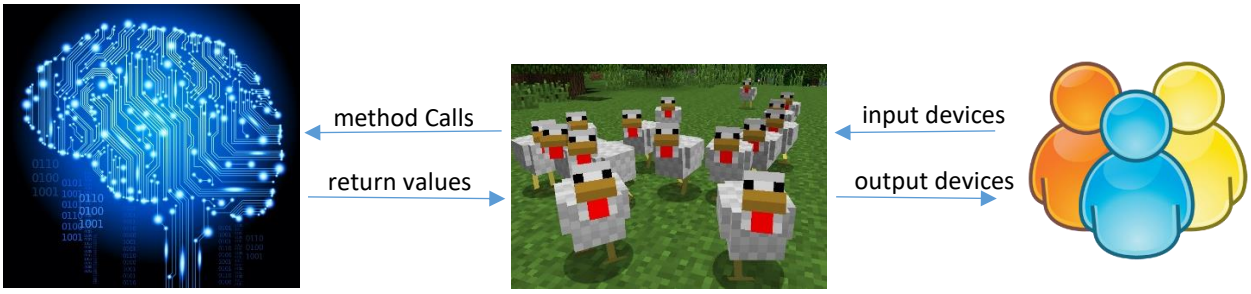
COMP10062: Assignment 2

The Assignment: The World of "ChickenCraft"

This assignment is about objects, instance variables, methods and encapsulation. You will create two classes to simulate the world of "ChickenCraft" – a simple game world loosely based on MineCraft Chicken objects.

In graphical applications, programmers often separate the **model** from the **view**. The model keeps track of the internal state of the program, and the view is in the middle between the user and the model. It talks to the user through a user interface, and it talks to the model by calling its methods and interpreting the return values from those methods. The model never talks directly to the user.

This is not a graphical application. In this assignment, the Chicken class is the model and the ChickenCraft class implements a "view" that consists of a text-based conversation with the user. If you implement the model well, it should be easy to re-use it later in a graphical view.



Step 1: Design and UML

Your first step should be to create a UML class diagram to represent a Chicken. You can create this diagram using UML software like UMLet or draw.io (links on Canvas). Please don't use a plain word processor like Microsoft Word. If you want to use a different piece of software, you must check with the instructor first.

- A Chicken can be happy or unhappy, and it can be alive or dead. A Chicken has a name, a certain number of "hearts" that represent its health, and a certain amount of seed in its stomach, in kilograms.
- You can feed a chicken some seed. Is hearts go up by 1 (maximum 4) every time you feed it. But don't feed it too much! A chicken with more than 2 kg of seed in its stomach will die.
- You can give a Chicken a different Chicken to play with. This makes both Chickens happy
- You can hit a Chicken and make its hearts go down by 1 (minimum 0). Hitting a Chicken always makes it unhappy and sometimes makes it dead (if it has 0 hearts).

- You can get eggs from a Chicken – you get one egg for every 0.25 kg of seed in its stomach (and the seeds used are gone from its stomach afterwards). If the Chicken is happy, you get twice as many eggs. But laying eggs makes a Chicken unhappy.
- Dead Chicken can't be fed, can't lay eggs, and can't play with other Chickens.
- When a Chicken is "born" (i.e. created) it alive and happy and has 4 hearts. By default, its name is "Nancy" and it has 0.1kg of seed, but you can change its name and the amount of seed it has after you create the chicken.

There should be a **toString** method that returns a full report on a Chicken. It is completely up to you what the return value of **toString** looks like, but don't just use the default from IntelliJ.

IMPORTANT BASIC RULE #1: The Chicken class is the **model**. The model should never talk to the user. It should do no input and produce no output.

Step 2: Implement the Chicken and ChickenCraft Classes

Once the class diagram is finished, implement your Chicken class in Java code. Then write a main method in a different class to simulate a user interacting with the Chickens in the world of ChickenCraft. This method should create three Chicken objects and allow a user to interact with each one using a menu interface. **It's up to you how you structure the dialog**, but one possibility is shown in the example dialog below.

```
1. Happy DEAD Chicken Syd: 2.1kg seeds, 4 hearts,
2. Happy Chicken Nancy: 0.6kg seeds, 4 hearts,
3. Happy Chicken Johnette: 1.0kg seeds, 4 hearts,

1. Feed    2. Play    3. Hit     4. Get Eggs     5. Quit
Choice: 4
Which chicken? 3
You got 8 eggs.

1. Happy Chicken Syd: 0.1kg seeds, 4 hearts,
2. Happy Chicken Nancy: 0.6kg seeds, 4 hearts,
3. Sad Chicken Johnette: 0.0kg seeds, 4 hearts,

1. Feed    2. Play    3. Hit     4. Get Eggs     5. Quit
Choice: 3
Which chicken? 2
Ouch!

1. Happy Chicken Syd: 0.1kg seeds, 4 hearts,
2. Sad Chicken Nancy: 0.6kg seeds, 3 hearts,
3. Sad Chicken Johnette: 0.0kg seeds, 4 hearts,

1. Feed    2. Play    3. Hit     4. Get Eggs     5. Quit
Choice: 4
Which chicken? 2
You got 2 eggs.
```

IMPORTANT BASIC RULE #2: The main method is the **view**. It talks to the user, calls the appropriate Chicken methods in response to the user's input, and displays the results. It does not determine the results of a user's actions or keep track of anything. That's the job of the Chicken objects.

Error Handling

If the user makes a mistake (feeds a bad amount of seeds, asks a chicken to play with a dead chicken, etc.) this should be reported to the user. But remember, the ChickenCraft class does not implement any logic to decide what happened, and the Chicken class is not allowed to use System.out.

Extra challenge: Implement your code in the `animate` method of the `FXAnimationTemplate` instead of in a `main` method. You will still get input from the user in the console, but you can show the Chickens on the canvas. See `GraphicsExampleForAssignment2.java` for an example of this way of using `animate`.

Encapsulation and Documentation

Make sure you **encapsulate** your instance variables. Only allow access through methods, using **get** and **set** method naming if appropriate. Don't forget the **toString** method, make sure you follow the **Documentation Standards** posted on Canvas.

Handing In

See the due date and time on the Canvas assignment. Hand in by attaching a zipped file of your two **.java** (not .class) files and your class diagram to the drop box.

Evaluation

Your assignment will be evaluated for performance (20%), class diagram (20%), structure (40%), and documentation (20%) using the rubric in the drop box.