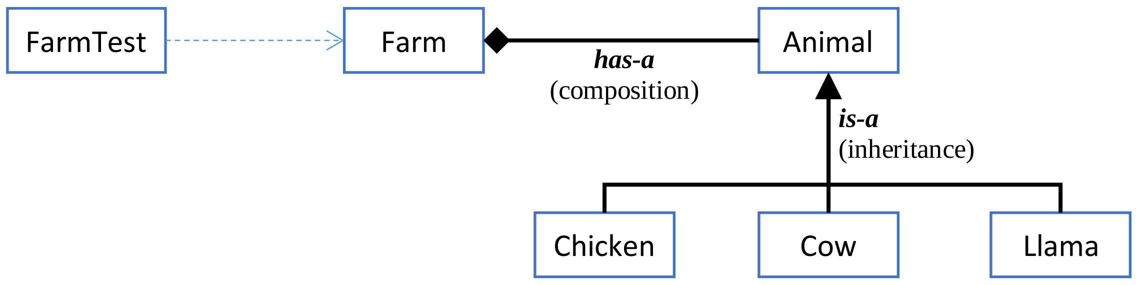
P1: Game Classes (42 marks)

***Focus:** basics of classes and objects, polymorphism*

In this part, you are required to create classes that represent our game objects. The UML design is given below followed by a brief description of each class.



[20 marks] **(A) `Animal` class** contains:

- [+1] Six private attributes:
  - `name` (`String`) - the animal's name.
  - `energy` (`double`) - the animal's energy level (must be from 0 to 100).
  - `alive` (`boolean`) - `true` only when `energy` > 0.
  - `mealAmount` (`double`) - the maximum amount of food the animal can eat when we call its `eat` method (must be from 0 to 100).
  - `x` and `y` (`double`) - the animal's location (no restrictions on the values).
  - `speedX` and `speedY` (`double`) - the speed in `x` and `y` directions (no restrictions on the values).
- [+1] One zero-arg constructor that sets `energy` to 100 and both, `speedX` and `speedY` to 1
- Several public methods:
  - [+2] Setters and getters for all attributes except `alive` which should have only a getter method.
  - Note the following:
    - [+2] your code must respect the restrictions on the values as indicated above.
    - [+2] The `energy`'s setter method should also display a message whenever the animal is hungry, e.g. "`Cow1 is hungry`", or starving, e.g. "`Cow1 is starving`", where Cow1 is the animal's name. An animal is starving when its `energy` ≤ 17 and it is hungry when its `energy` is between 17 and 50.
  - [+2] **`toString`** method that returns a string representation of the animal's status similar to the following format:

```
Cow1: alive at (0.0,0.0) Energy=100.0
```

  - [+2] **`void move()`**: if the animal is alive, increment `x` by `speedX` and `y` by `speedY` , and decrement energy by 0.1. If the animal is dead, display a message to indicate that it is dead, e.g. "`Cow1 can't move. It is dead!`"

- [+2] **`void speak(String msg)`** which displays the given `msg` on the console only if the animal is alive, e.g. "`Cow1 says: Hi.`"
- [+2] **`void sound()`**: if the animal is alive, display on the console "`Unknown sound`".
- [+4] **`double eat()`** which increments `energy` according to the following rules:
  - An animal can only eat if it is alive and not full. If it is dead, display a message such as "`Cow1 is dead!`". If it is full, display a message such as "`Cow1 didn't eat as it is full!`"
  - When we call eat(), the animal will eats a maximum of `mealAmount`. For example, if Cow1's `mealAmount` is 20 and its `energy` is 60, then the amount it eats after calling the `eat()` method will be 20 and its energy will be 80. On the other hand, if Cow1's `energy` is 95, it will eat only 5 units, bringing its energy to 100.
  - The `eat()` method returns whatever amount of food the animal eats. This will be deducted from the amount of available food on the farm (will discuss this shortly).
  - The `eat()` method should print on the console the amount eaten, e.g. `Cow1 ate 5 units, now it is full!`

[10 marks] **(B) `Cow`, `Chicken`, and `Llama` classes: in each class, do the following:**

- [+3] Override the **`sound()`** method in order to display: "Moo" for Cows, "Hmmm!" for Llamas, and "Cluck!" for chicken. An animal can make a sound only if it is alive.
- [+3] Add one more method to each class: void `swim()` to `Chicken`, `Jump()` to `Llama`, and `Milk()` to `Cow`. Each method should display some text about how the animal can perform these actions (few words should do).
- Add a zero-arg constructor that:
  - [+1] Sets `mealAmount` to: 20 for `Cow`, 9 for `Llama`, and 5 for `Chicken`.
  - [+3] Automatically assigns a name to every new object that is composed of the class name followed by a number that is incremented automatically for every new object. For example, assume you create 5 animals in this order: 2 cows, a chicken, another cow, and another chicken. Your code should automatically give the first two cows the names `Cow1` and `Cow2`, then the name `Chicken1` to the first chicken, then name `Cow3` to the third cow, and finally the name `Chicken2` to the second chicken.

[10 marks] **(C) the `Farm` class** contains:

- [+1] An array `animals` which when initialized will hold a list of 4 animals: a `Chicken`, a `Cow`, and 2 `Llamas`.
- [+1] A private `double` attribute `availableFood` which represents the amount of food available in the farm for the animals (must be from 0 to 1000).
- [+2] A zero-arg constructor that sets `availableFood` to 1000 and initializes `animals`.
- [+2] A public method `void makeNoise()` that causes all animals to `sound()`.

- [+2] A public method `void feedAnmials()` that iterates over each animal, calls its `eat()` method, and then decrements `availableFood` by the amount eaten. Once the availableFood is not enough to feed the animal, your code should display a message that there is not enough food in the farm.
- [+2] Setters and getters for all attributes except `animals` which should have only a getter method.

[2 marks] **(D) `FarmTest` class** is used solely to test other classes and ensure the code works properly. For example, copy the code below to your program and run it:

```
public class FarmTest {
   public static void main(String[] args) {
       Farm myFarm = new Farm();
       for(Animal a: myFarm.getAnimals())//move each animal by random amount
          for(int i=0; i<(int)(Math.random()*1000); i++) a.move();
       System.out.println("Before feeding: " + myFarm.getAvailableFood());
       myFarm.feedAnimals();
       System.out.println("After feeding: " + myFarm.getAvailableFood());
       myFarm.makeNoise();
       myFarm.getAnimals()[0].speak("Hello!!");
       System.out.println(Arrays.toString(myFarm.getAnimals()));
   }
}
```

Above code should print an output similar to the one below. Note that this output may be slightly different based on several factors such as your animals' energy and the available food in the farm.

```
Before feeding: 1000.0
Chicken1 ate 5.0 units
Cow1 ate 0.3 units. Now it is full!
Llama1 ate 5.5 units. Now it is full!
Llama2 ate 1.8 units. Now it is full!
After feeding: 987.4
Cluck!
Moo!
Hmmm!
Hmmm!
Chicken1 says: Hello!!
[Chicken1: alive at (50.0,50.0) Energy=95.0  ,
 Cow1    : alive at (3.0,3.0) Energy=100.0  ,
 Llama1  : alive at (55.0,55.0) Energy=100.0  ,
 Llama2  : alive at (18.0,18.0) Energy=100.0  ]
```

Note: the last four lines will actually be printed on the same line on the console, but I put them here on four separate lines for clarity. Also, double values may have minor precision error on your machine (e.g. 987.4 may display as 987.400000000001).

Feel free to change the code in `FarmTest` to test other methods in your classes. You will get the full mark for this part (i.e. part (D)) if you just try the code given above and it works.

Submission Instructions

For this part of the project, you need to do the following:

1- Create a Java project of with any name of your choice.
2- Create a package with the name P1 and write your code within this package.
3- Zip the package P1 and name the zipped file as: YourStudentID_P1. e.g., “1234567_P1”.
4- Submit the zipped file to Canvas.

Note that you can resubmit an assignment one more time, but the new submission overwrites the old submission and receives a new timestamp.