

# CS1020: DATA STRUCTURES AND ALGORITHMS I

## Tutorial 4 – Inheritance, Polymorphism

(Week 6, starting 15 February 2016)

### 1. Inheritance

New McDonald has a farm with some animals.

Each animal has a name, and makes a sound. Some animals are flyers. These animals can fly – once they start to fly, their sound is “flap” till they stop. Some flyers are gliders. These animals can glide – if they are flying, once they start to glide, their movement is “whoosh” till they stop.

Your junior has written this code to describe the animals in the farm. However, his code cannot be compiled, and will not work properly. There are 3 errors within the 3 classes; 1 of them affects compilation.

Tip: Try to compile your code in sunfire! Once the compilation error is rectified, instantiate objects and test.

```
class Animal {
    private String _name; // e.g. Cow
    private String _sound; // e.g. Moo
    public Animal(String name, String sound) {
        _name = name; _sound = sound; }
    public String getName() { return _name; }
    public void makeSound() {
        System.out.println(_name + " goes " + _sound); }
}

class Flyer extends Animal {
    protected String _name;
    protected String _sound;
    protected boolean _isFlying;
    public Flyer(String name, String sound) { Animal(name, sound); }
    public void makeSound() {
        if (_isFlying)
            System.out.println(getName() + " goes flap");
        else super.makeSound();
    }
    public void fly() { _isFlying = true; }
    public void stop() { _isFlying = false; }
}

class Glider extends Flyer {
    private boolean _isGliding;
    public Glider(String name, String sound) { super(name, sound); }
    public void glide() { if (_isFlying) _isGliding = true; }
    public void stop() { _isFlying = false; _isGliding = false; }
    public void makeSound() {
        if (_isGliding)
            System.out.println(getName() + " goes whoosh");
        else makeSound();
    }
}
```

- (a) What does the **protected** keyword mean? In this example, how is it useful?
- (b) Within a method in the Flyer class, why can getName( ) be invoked?
- (c) How is overriding demonstrated here, and how is it useful?
- (d) Identify and rectify the 3 errors.

## 2. Inheritance & Polymorphism

Related to T2Q2 and T4Q1, New McDonald still has a farm with some animals. You want to **make use of polymorphism** to allow 5 animals in the farm to makeSound( ), without having to concern yourself about what type of Animal each is. To simplify things, let's ignore Gliders. There are only Flyers and non-Flyers.



**Parrot**  
"squak"  
Perched



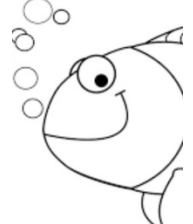
**Cow**  
"moo"



**Mosquito**  
"buzz"  
Flying



**Sheep**  
"mehh"



**Fish**  
"blurp"

Picture credits: clipartpanda.com

Some code is already given to you. Answer the questions on the next page.

```
class Animal { ... } // As in Q1
class Flyer extends Animal { ... } // As in Q1

public class NewMcDonald {
    private Animal[] _farm; // New McDonald had a farm (still has now)
    private static final int SIZE = 5; // Fixed farm size of 5

    public NewMcDonald() {
        /* TODO: Create your farm, an Animal array */
    }

    public void makeSomeNoise() {
        /* TODO: Make sound(s) without looking out for Flyers...! */
    }

    public void fillThisFarm() {
        _farm[0] = new Flyer("Parrot", "squak"); // (a)
        _farm[1] = new Animal("Cow", "moo");
        _farm[2] = new Flyer("Mosquito", "buzz");
        ((Flyer)_farm[2]).fly(); // (b)
        _farm[3] = new Animal("Sheep", "mehh");
        _farm[4] = new Animal("Fish", "blurp");
    }
}
```

- (a) What is the data type of `_farm[0]`? Why can a `Flyer` object be assigned to `_farm[0]`?
- (b) Why can't `((Flyer)_farm[2]).fly()` be replaced with `_farm[2].fly()`?
- (c) Complete the constructor and `makeSomeNoise()` method. The output of `makeSomeNoise()` is:
- ```
Parrot goes squak
Cow goes moo
Mosquito goes flap
Sheep goes mehh
Fish goes blurp
```

*Tip:* If you are unsure of your answers, **try out** the program, and ensure you **revisit lecture** concepts!

### 3. Enhancing Data Structures

You are given a generic `Pair<T, S>` class. Each object of this class can refer to 2 objects of different types:

```
class Pair <T, S> {
    private T _objLeft;
    private S _objRight;

    public Pair(T pObjLeft, S pObjRight) {
        _objLeft = pObjLeft;
        _objRight = pObjRight;
    }

    public T getLeft() { return _objLeft; }
    public S getRight() { return _objRight; }
}
```

We now want to create a `GenericTriple<T, U, V>` class. Each object of this class can refer to 3 objects of different types. Restriction: For each part, you may only have **ONE** attribute. There are 2 ways to achieve this:

- (a) Use inheritance.
- (b) Use composition. `GenericTriple` is composed of a `Pair`, i.e. it will have a reference to a `Pair` object as an attribute, and the `Pair` object only lives as long as the `GenericTriple` lives.  
*[Hint: How do you store 3 objects in a `Pair`? Use 2 `Pair` objects.]*

We can now instantiate a `GenericTriple` object that refers to 3 objects. A `Person` has a name (`String`), weight (`double`), and height (`double`). Create a `Person` data structure and use a `GenericTriple` to help you store data:

- (c) Use inheritance. *[Hint: Are you inheriting a family of classes, or just one specific type?]*
- (d) Use composition. `Person` is composed of a `GenericTriple`.

*Tip:* Don't test your brain power!

**Draw** out how each data structure should look like in memory **BEFORE** coding.

- You are now halfway through 😊 -

Have you been revising, learning,  
and **\*practicing\*** daily?