CSC240_Lab04: Inheritance and Polymorphism

Points: **100** points.

Background:

In this lab, we will use the well-known song 'Old MacDonald Had a Farm' to learn about **`Inheritance`** and **`Polymorphism`**.

Old MacDonald had a farm and several types of animals. Every animal shared certain characteristics. They each had a type (such as cow, chick or dog) and each made a sound (moo, cheap or oink). An Interface defines those things required to be an animal on the farm.

```
public interface Animal
{
  public String getSound();
  public String getType();
}
```

Notes:

For those unfamiliar with it, a version of the *Old MacDonald* song is found at http://www.scoutsongs.com/lyrics/oldmacdonald.html.

And its video at: https://www.youtube.com/watch?v=nFX98pqzb3o

Assignment:

1. Once we know what it takes to be an `Animal`, we can define new classes for the cow, chick and dog that implement the `Animal` interface. Here is a `Cow` class meeting the minimum requirements to be an `Animal`.

```
public class Cow implements Animal
{
  private String myType;
  private String mySound;

  public Cow()
  {
    myType = "cow";
    mySound = "moo";
  }

  public String getSound()
  {
    return mySound;
  }

  public String getType()
  {
    return myType;
  }
}
```

2. Implement classes for the chick and the dog. Also complete the test program below to verify your work so far:

```
public class Tester
{
  public static void main(String[] args)
  {
    Cow c = new Cow();
    System.out.println( c.getType() + " goes " + c.getSound() );

    // < your code here >
  }
}
```

3. Create a complete farm to test all your animals. Here is the *Farm.java* source code.

```
public class Farm
{
  private Animal [] myFarm;

  public Farm()
  {
    myFarm = new Animal [3];
    myFarm[0] = new Cow();
    myFarm[1] = new Chick();
    myFarm[2] = new Dog();
  }

  public void animalSounds()
  {
    Animal temp;
    for (int i = 0; i < myFarm.length; i++)
    {
      temp = myFarm[i];
      System.out.println(temp.getType() + " goes " + temp.getSound());
    }
  }
}
```

You will need to change your *OldMacDonald.java* code to create an object of type `Farm` and then to invoke its `animalSounds` method.

4. As it turns out, the chick seems a little confused. Sometimes it makes one sound, when it is feeling childish, and another when it is feeling more grown up. Its two sounds are "cheep" and "cluck". Modify the *Chick.java* code to add a second constructor that sets a flag for the chick to indicate whether the chick makes one or two sounds. The `getSound()` method returns either sound, with equal probability, if there are two sounds available. You will also have to modify your *Farm.java* code to construct the `Chick` with two possible sounds.

5. Finally, it also came to pass that the cows received personal names, such as Elsie. Create a new class, `NamedCow`, which extends the `Cow` class, adding a constructor, a field for the `Cow's` name, and a new method: `getName`. Shown below is the final *Farm.java* code to exercise all your modifications:

```
public class Farm
{
   private Animal [] myFarm;

   public Farm()
   {
      myFarm = new Animal [4];
      myFarm[0] = new Cow();
      myFarm[1] = new Chick();
      myFarm[2] = new Dog();
      myFarm[3] = new NamedCow("Elsie");
   }

   public void animalSounds()
   {
      Animal temp;
      for(int i = 0; i < myFarm.length; i++)
      {
         temp = myFarm[i];
         System.out.println(temp.getType() + " goes " + temp.getSound());
      }

      Chick tweety = new Chick(true);
      for(int i = 0; i < 5; i++)
      {
         System.out.println(tweety.getSound());
      }

      NamedCow named = (NamedCow)myFarm[3];
      System.out.println(named.getName());
   }
}
```

6. Make sure that you understand what you've just accomplished. Having an array of `Animal` objects and then having the `getSound()` method dynamically decide what sound to make is known as *polymorphism*. This is also known as *late binding* because it wasn't known until run-time that `a[1]`, for example, really was a `Chick` object.

You started with an *Interface* for an `Animal` and then used the keyword **`implements`** in making the three types of animals. Then you created a specialized version of the `Cow`, a `NamedCow`, using the keyword **`extends`**. This illustrates the concept of inheritance. The `NamedCow` had all the attributes and methods of the `Cow` and then added some: a new field and a new method to access the cow's name.

Instructions:

1. Develop and test the Old MacDonald Farm classes as described in the Assignment section above.
2. Your lab assignment should have a zipped project that consists of the following 7 files:

   Animal.java - interface
   Chick.java, *Cow.java*, *Dog.java* - implementations of the Animal interface
   NamedCow.java - subclass of the Cow class
   Farm.java - collection of Animal objects
   OldMacDonald.java - testing class

3. Upload your zipped project source code and run output through D2L.