CS1010J Programming Methodology
Tutorial 11: Basic OOP Part III

Programming is fun, no kidding!

~ Dr. Zhou Lifeng

To students:

Last tutorial! ☺

I. Manual Tracing

We have written the following **Point** class in Tutorial 8. It will be used in Q1 and Q2 below.

```
class Point {

  private double x, y;  // x- and y- coordinates

  // Construct a point of coordinates (inX, inY)
  public Point(double inX, double inY) {
    x = inX;
    y = inY;
  }


  // Compute the distance between
  // 'this' point and 'another' point
  public double distanceTo(Point another) {

    double xDiff = x - another.x;
    double yDiff = y - another.y;

    return Math.sqrt( xDiff*xDiff, + yDiff*yDiff );
  }


  // Return the string representation
  // of a point in the format "[x, y]"
  public String toString() {
    return "[" + x + ", " + y + "]";
  }
}
```

1. The **Point** class represents a point in a 2-dimensional plane.

   (a) Add a **member method** **`lessThan()`** to the **Point** class definition. This method returns true if 'this' point is smaller than 'another' point, or false otherwise.

   A point is smaller than another if its x-coordinate if smaller, or in the case x-coordinate is the same, its y-coordinate is smaller.

```
class Point {

  // Other code omitted for brevity

  // Return true if 'this' point is
  // less than 'another', or false otherwise.
  public boolean lessThan(Point another) {

    /* fill in your code below */

     if (this.x < another.x){
       return true;
     }
     else if (this.x == another.x && this.y < another.y){
       return true;
     }
  }  else{
}      return false;
```

   (b) Given the **Point** class definition, write a **user method** **`smallest`**() that takes an array of **Point** objects and returns the index of the smallest point. In case there are multiple smallest points, return the index of the first one in the array.

```
// Return index of the first smallest point
public static int smallest(Point[] points) {

  /* fill in your code below */

   int smallestIdx = 0;
   for (int i = 0; i < points.length; i++){
     if (!points[smallestIdx].lessThan(points[i])){
     smallestIdx = i;
   }

   }




}
```

(c) Write a **user method** **`createPoints()`** that creates an array of 10 points whereby the 1st point has coordinates (0, 0), the 2nd point has coordinates (0, 1), … the 10th point has coordinates (0, 9). This method returns the array created.

```
// Create an array of 10 points
public static Point[] createPoints() {

  /* fill in your code below */

  Point[] point = new Point[10];
  for (int i = 0; i < 10; i++){
    point[i] = new Point(0, i);
  }




}
```

(d) What's the output of the following **user method**?

```
public static void main(String[] args) {

  Point[] points = createPoints();

  for (int i = 0; i < points.length; i++) {
    System.out.println(points[i]);
  }
                    prints all the points in the format [x, y]

  System.out.println("Smallest point = " +
                     points[ smallest(points) ] );
}
```
[0, 0]

2. Given the **Point** class definition, let's write another class **Line** that represents a line using its two endpoints.

```
class Line {

  Point end1, end2;

  // Create a line with p1 and p2 as endpoints
  public Line(Point p1, Point p2) {
    /* fill in your code below */
     this.end1 = p1;
     this.end2 = p2;

  }

  // Return the length of this line
  public double getLength() {
    /* fill in your code below */
    return Math.sqrt(Math.pow((end2.x - end1.x), 2) + Math.pow(end2.y - end1.y), 2))
     return this.end1.distanceTo(this.end2); //composition Line has-a Point object



  }
}
```

(a) Fill in the **Line** class above for the parts marked with /* fill in your code below */.

(b) Given the **Line** class, what's the output of the following **user method**?

```
public static void main(String[] args) {

  Point point1 = new Point(3, 0);
  Point point2 = new Point(3, 4);
  Line  line1  = new Line(point1, point2);
  Line  line2  = new Line(new Point(0, 0), point2);

  System.out.println(line1.getLength());
  System.out.println(line2.getLength());
}
```

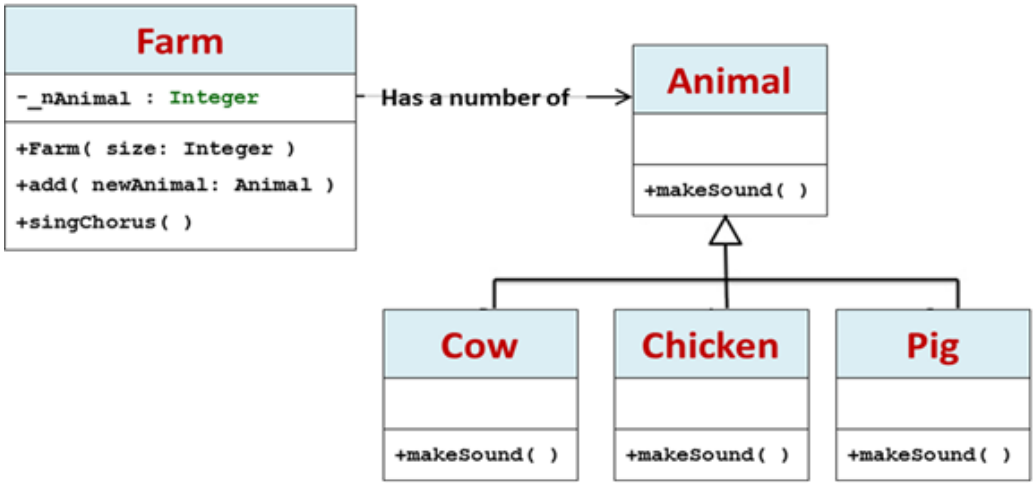
To students:

We may create an anonymous object (as demonstrated when creating **`line2`**) when there is no need to give it a name. Such an object is often for "one-time" use only.

3. There is a nursery rhyme “Old McDonald” that goes like this:

> **Old McDonald had a farm, E-Ya-E-Ya-O**
> **And, on his farm he has some Cows, E-Ya-E-Ya-O**
> **There is a "Moo Moo" here, and a "Moo Moo" there**
> fd
> **……**
> **Old McDonald had a farm, E-Ya-E-Ya-O**
> **And, on his farm he has some Chicks, E-Ya-E-Ya-O**
> **There is a "Pook Pook" here, and a "Pook Pook" there**
> **…….**
> **Old McDonald had a farm, E-Ya-E-Ya-O**
> **And, on his farm he has some Pigs, E-Ya-E-Ya-O**
> **There is a "Onik Onik" here, and a "Onik Onik " there**
> **…….**

As we can see, a **Farm** stores a number of **Cow**, **Chick** and **Pig** objects with their relationships shown as follows:



The above diagram suggests that the best design is to create an **Animal** class and use inheritance and polymorphism to model relations of animals. When the **`singChorus()`** method of the **Farm** class is invoked, all animals on the farm will make sounds in turn.

We will not discuss inheritance and polymorphism in CS1010J and therefore will take a simpler design: **Farm** class directly keeps the following three animals as attributes: a **Cow**, a **Chicken** and a **Pig**.

(a) Fill in the 4 classes below for the parts marked with /* fill in your code below */.

```
class Cow {
  // Cow sings
  public void makeSound() {
    System.out.println("Moo Moo");
  }
}

class Chicken {
  // Chicken sings
  public void makeSound() {
    /* fill in your code below */
     System.out.println("Pook Pook");
  }
}

class Pig {
  // Pig sings
  public void makeSound() {
    /* fill in your code below */
     System.out.println("Oink Oink");
  }
}

class Farm {

  private Cow cow;
  private Chicken chick;
  private Pig pig;

  public Farm(Cow newCow, Chicken newChick, Pig newPig) {
    /* fill in your code below */
     this.cow = newCow;
     this.chick = newChick;
     this.pig = newPig;
  }

  public void singChorus() {
    /* fill in your code below */




  }
}
```

(b) What's the output of the following **user method**?

```
public static void main(String[] args) {

  Cow cow = new Cow();
  Chicken chicken = new Chicken();
  Pig pig = new Pig();

  Farm farm = new Farm(cow, chicken, pig);
  farm.singChorus();
}
```

To students:

Two versions of the programs are provided for your reference: one use inheritance and polymorphism and the other not. Inheritance and polymorphism are not examinable in CS1010J and will be formally introduced in CS2030.

II. Programming

4. **[Problem Set 4 Exercise #29] Balls**