Experiment # 21: Abstract class

Write a Java program to create a class called "Shape" with abstract methods for calculating area and perimeter, and subclasses for "Rectangle", "Circle", and "Triangle".

Write a Main class that creates instances of different shape objects. It then calls methods to calculate the area and perimeter of each shape and displays the results in the console.

Hint: Rectangle (**Area** = $length * width$, **Perimeter** = 2*($length + width$))

Circle (**Area** = $\pi * radius^2$, **Perimeter** = $2\pi * radius$)

Triangle (**Area** = $\sqrt{s * (s - side1) * (s - side2) * (s - side3)}$ where $s = (side1 + side2 + side3)/2$, **Perimeter** = $side1 + side2 + side3$)

```
package Lab_8;


abstract class Shape{
   abstract double calculateArea();
   abstract double calculatePerimeter();

}
```

```
package Lab_8;


class Rectangle extends Shape {
   private double length;
   private double width;

   public Rectangle (double legth, double width)
   {
      this.length = legth;
```

```
        this.width = width;
    }

    @Override
    double calculateArea() {
        return length*width;
    }

    @Override
    double calculatePerimeter() {
        return 2*(length + width);
    }
}

package Lab_8;

class Circle extends Shape {
    private double radius;

    public Circle(double radius){
        this.radius = radius;
    }

    @Override
    double calculateArea(){
```

```
        return Math.PI*radius*radius;
    }

    @Override
    double calculatePerimeter(){
        return Math.PI*2*radius;
    }

}



package Lab_8;

class Triangle extends Shape {
    private double side1;
    private double side2;
    private double side3;

    public Triangle (double side1, double side2, double side3){
        this.side1 = side1;
        this.side2 = side2;
        this.side3 = side3;
    }

    @Override
```

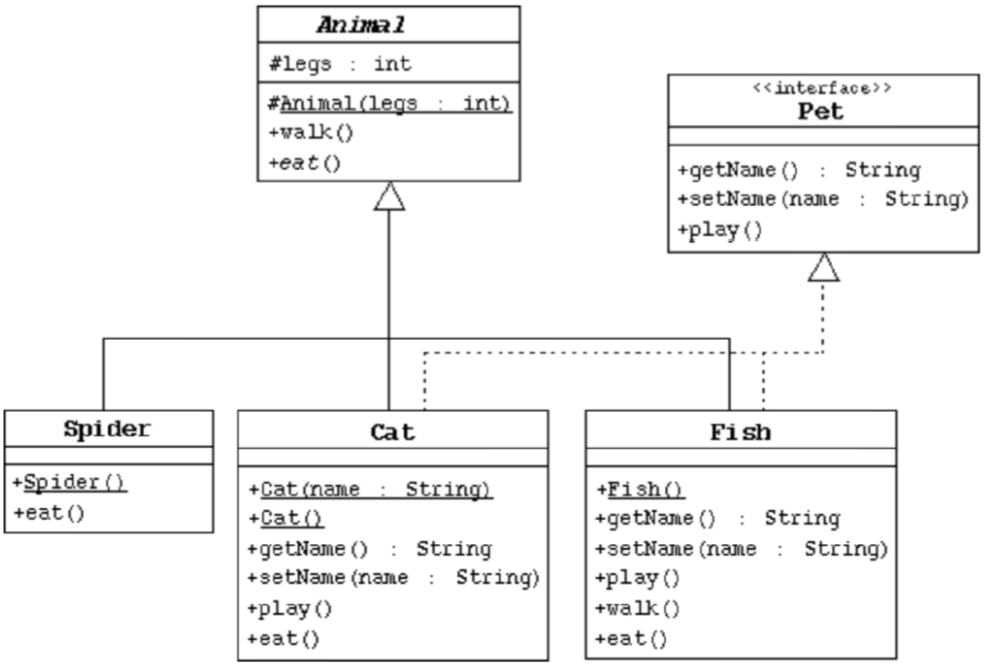
```
    double calculateArea() {
        double s = (side1 + side2 + side3)/2;
        return Math.sqrt(s*(s-side1)*(s-side2)*(s-side3));
    }

    @Override
    double calculatePerimeter(){
        return side1 + side2 + side3;
    }
}
package Lab_8;

public class Main {
    public static void main(String[] args) {
        Shape rectangle = new Rectangle(5,10);
        Shape circle = new Circle(7);
        Shape triangle = new Triangle(3, 4, 5);

        System.out.println("Rectangle Area:" +rectangle.calculateArea());
        System.out.println("Rectangle Perimeter:" +rectangle.calculatePerimeter());
        System.out.println("Circle Area: " + circle.calculateArea());
        System.out.println("Circle Perimeter: " +circle.calculatePerimeter());
        System.out.println("Triangle Area: " +triangle.calculateArea());
        System.out.println("Triangle Perimeter: " +triangle.calculatePerimeter());
```

```
    }
}
```

Experiment # 22: Using Interfaces and Abstract Classes

Create a java code for the following UML diagram (*hint: abstract classes and methods are italicized*).

Create a main method to experiment different variations of these animals, their methods, and polymorphism.




```
package Lab_8;


public abstract class Animal {

   protected int legs;
```

```
    protected Animal(int legs){
        this.legs = legs;
    }

    public abstract void eat();

    public void walk(){
        System.out.println("This animal walks with" +legs +"legs.");
    }
}
```

```
package Lab_8;

public interface Pet {
    String getName();
    void setName(String name);
    void play();
}
```

```
package Lab_8;

public class Spider extends Animal {
    public Spider(){
        super(8);
```

```
    }

    @Override
    public void eat(){
        System.out.println("The spider eats indects");
    }
}

}
package Lab_8;

public class Cat extends Animal implements Pet {

    private String name;

    public Cat(String name){
        super(4);
        this.name = name;
    }

    @Override
    public String getName(){
        return name;
    }
```

```
    @Override
    public void setName(String name){
        this.name = name;
    }

    @Override
    public void play(){
        System.out.println(name + "is playing with a ball.");
    }

    @Override
    public void eat(){
        System.out.println(name +"is eating cat food.");
    }

}
package Lab_8;

public class Fish extends Animal implements Pet {
    private String name;

    public Fish(){
        super(0);
    }
```

```
    @Override
    public String getName(){
        return name;
    }

    public void setName(String name){
        this.name = name;
    }

    @Override
    public void play(){
        System.out.println(name + "is swimming around playfully.");
    }

    @Override
    public void walk(){
        System.out.println("Fish can't walk but swims");
    }

    @Override
    public void eat(){
        System.out.println(name + " is eating fish food.");
    }
}
```

```
package Lab_8;


public class TestAnimals {
   public static void main(String[] args) {
      Animal spider = new Spider();
      spider.walk();
      spider.eat();


      Pet cat = new Cat("Whiskers");
      cat.play();
      System.out.println("Cat's Name: " + cat.getName());
      cat.setName("Fluffy");
      System.out.println("Cat's new Name" + cat.getName());


      Pet fish = new Fish();
      fish.setName("Goldy");
      fish.play();
      System.out.println("Fish's name: " +fish.getName());
      ((Fish)fish).walk();
      ((Fish)fish).eat();
   }
}
```